Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung (ITIV)

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker (Sprecher)
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Bachelorarbeit Nr. ID-3509

von Herrn cand. wing. Marcel **Namyslo**

ATHENA: Advanced Learning through Efficient Transparent Model Harnessing with Explainable Data

Beginn:	22.07.2024
Abgabe:	06.12.2024
Betreuer:	M.Sc. Moritz Zink Institut für Technik der Informationsverarbeitung M.Sc. Daniel Grimm Institut für Technik der Informationsverarbeitung
Korreferent:	Prof. Dr.rer.nat. Stefan Nickel Institut für Operations Research (IOR)

Hauptreferent: Prof. Dr.-Ing. Eric Sax

I declare that I have developed and written the enclosed thesis completely by myself. I have submitted neither parts of nor the complete thesis as an examination elsewhere. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. This also applies to figures, sketches, images and similar depictions, as well as sources from the internet.

Karlsruhe, November 13, 2025

Marcel Namyslo

Abstract

Artificial neural networks have become increasingly popular for image classification tasks [1]. They have proven remarkable performance in high-dimensional data, making them a popular tool in various sophisticated applications. Backed by the growing combined CPU and GPU power, these models can now be trained on millions of samples [1]. Together with that, new approaches are published every year to improve and speed up the training process and decrease the computational effort [2]. These approaches include a variety of methods, ranging from modifying training data and optimizing training processes to altering model structures. In this thesis, a novel approach will be introduced to boost the computational power of model algorithms.

Explainable Artificial Intelligence (XAI) is an emerging area of research in the field of Artificial Intelligence [3]. Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy. However, the highest accuracy for large modern datasets is often achieved by models with complex architecture, ultimately leading to significant computational effort. To change the original purpose of XAI of interpreting and explaining model decisions, new approaches recently have been introduced to utilize XAI to leverage up the training model [4][5]. The desired benefits of these methods are more accurate models, as well as higher efficiency in the model's training.

This thesis will demonstrate a new approach that leverages XAI to train deep neural networks more efficiently. Therefore, different established XAI methods will be reviewed and compared until the most suitable and promising algorithm is found for implementation. Finally, this workflow aims to create models that are not only more efficient but also maintain or even improve their accuracy.

Contents

1 Introduction

1.1 Motivation

Artificial Intelligence (AI) has taken a huge role across many industries and its application can be found in various fields, such as autonomous driving, medicine or agriculture [6]. This trend of adoption and integration of AI technologies continues to expand rapidly, and simultaneously it is becoming increasingly important to ensure these systems are not only effective but also transparent, understandable, and secure [3].

Unfortunately, AI models often operate as "black boxes". While we know the inputs and outputs, the processes driving their decisions remain hidden [7]. This lack of transparency stems from the complex and high-dimensional nature of these models, making it feel almost impossible to fully understand how they arrive at their conclusions. Yet, understanding how AI makes decisions is crucial for building trust in its predictions, especially in high-stakes areas like medical diagnostics, autonomous navigation, and financial risk assessment [8]. A single misclassification or incorrect prediction can lead to serious consequences. This is a huge problem, as the lack of transparency raises concerns about the reliability and accountability of AI. Especially as it takes on roles where safety, fairness, and trust are paramount.

To address these challenges, the field of Explainable AI (XAI) has emerged. XAI methods enable us to analyse and interpret the decision-making processes within a model and make the importance of various influencing factors visible[9]. If we uncover what drives a model's decisions, we can validate the decision and diagnose weaknesses, which ultimately leads to a more reliable model.

However, XAI has the potential to extend beyond the purpose of explanation. A promising new direction is emerging which includes using XAI not only for understanding models but for enhancing their performance [4][5]. By integrating XAI into the training process itself XAI can enhance efficiency by reducing training time or the amount of data required. This innovative approach broadens XAI's role from purely explanatory to actively improving model training.

To see how this idea could be applied, we can look at a computer vision model for an autonomous driving system. Training such a model to recognize road signs, pedestrians, and other critical elements typically requires an enormous amount of data. This process is not only resource-intensive but also highly time-consuming.

On top of that, achieving optimal accuracy often involves multiple rounds of training and fine-tuning, further increasing development time and computational costs.

Given these challenges, finding ways to make the training process more efficient is incredibly valuable. By improving training methods, models could achieve high accuracy with less data and fewer iterations which saves both time and resources.

1.2 Objectives

This thesis explores how XAI can be utilized to enhance the efficiency of AI training processes. It compares various XAI methods and approaches, investigates how they can be applied to improve training efficiency, and demonstrates a concrete implementation. The primary goal is to integrate XAI into the training process to improve model accuracy while using less amount of training data.

1.3 Structure of thesis

To select the most suitable XAI method for efficient retraining, it is essential to first understand the fundamentals of computer vision and machine learning. Chapter 2 begins with an introduction to the basics of deep neural networks and their application to computer vision tasks. We will then explore the challenges and limitations of DNNs that have driven research into the field of XAI, followed by a discussion of the principles and ideas behind explainable AI.
In chapter 3, we will present a selection of the most promising and widely recognized XAI methods, explaining their concepts, advantages, and disadvantages. Next, in Chapter 4 we will present existing approaches to leverage XAI for efficient model training and evaluate, which of the previously discussed methods is the most suitable to use xAI for efficient model training. In chapter 5, we present the current implementations of these approaches, after which we move forward to chapter 6, where we explore how the selected method can be integrated into model training. Finally, chapter 7 will discuss the results, after which we conclude this thesis in chapter 8 with our conclusion and future work.

2 Fundamentals

This chapter explains the theoretical fundamentals for this work. First, a short introduction to the functionality of Neural Networks will be provided to help understand how the XAI methods work with them. Then a strong focus will be put on the principles of explainable AI.

2.1 Deep Neural Networks in Computer Vision

Computer vision tasks can be classified into three levels based on their complexity and abstraction, as demonstrated in Figure 2.1. This work focuses specifically on the field of object classification and the application of XAI to it. Figure 2.2 emphasizes the difference between the three most common high-level vision tasks, classification, object detection, and segmentation. In machine learning, classification simply means to assign a label to an input from a predefined set of categories, whereas object detection involves localizing the objects in an image, and instance segmentation involves directly detecting the shapes of the objects [10].

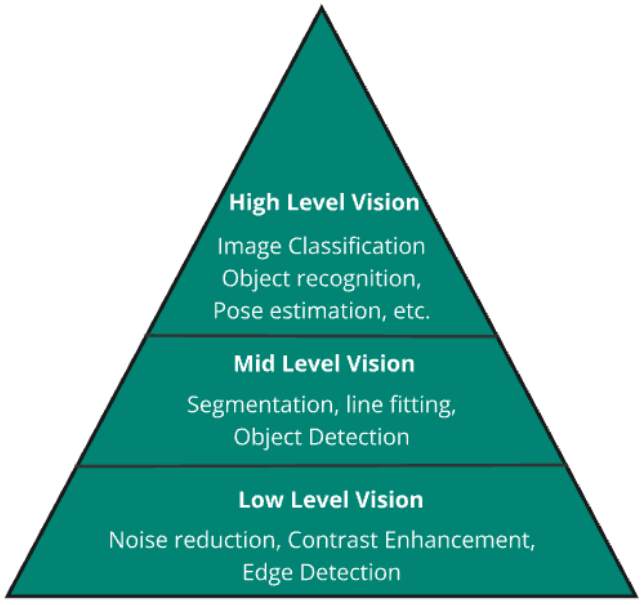

Figure 2.1: Computer Vision Tasks [11]

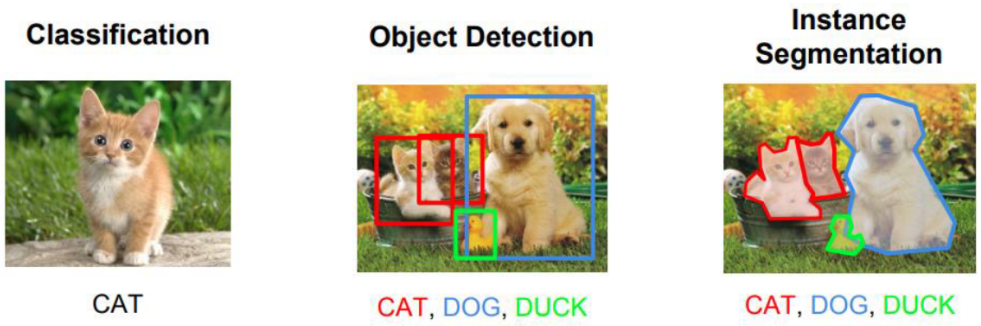

Figure 2.2: Traditional Machine Learning [12]

The key challenge in machine learning and AI is to empower systems to make decisions driven by data insights [13]. In order to create an accurate AI model, an enormous amount of data is necessary to train the model so it can predict data confidently. However, many real-world datasets, particularly images, speech, and text, exhibit complex, nonlinear patterns. This is problematic for traditional Machine Learning Models such as linear regression, logistic regression, and decision trees (Figure 2.3) due to their assumption and the necessity of smooth linear data [14] [15]. Linear data specifically means that there is a measurable proportionality between the data points.

Figure 2.3: Traditional Machine Learning Models

2.1.1 Linear Networks

To recognize and comprehend these complex patterns inside the input data, a shift happened from traditional machine learning to deep learning and deep neural networks [16]. A fundamental part of DNNs are fully connected linear networks, which are characterized by their sophisticated network structure and which are used to gain knowledge from high-level data. They proved that the deeper the structure of a model and thus the more complex a model gets, the more precisely and accurately it can perform on more difficult tasks [17].

The network's predecessor was published in 1958 when Frank Rosenblatt [18] introduced his idea of the Perceptron. It was a binary classifier and worked as a threshold function where positive outputs were assigned to one class while negative ones were assigned to the other class. This work was then further developed by Minsky [19], who introduced the concept of numerical weights for inputs as seen in Figure 2.4. These weights help determine which input has more importance for the output.

However, a single binary Perceptron has its limitations, particularly its inability to solve nonlinear classification problems [20]. To overcome these flaws, multi-layer deep neural networks were introduced.

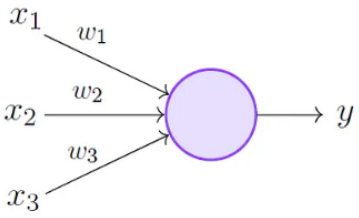

Figure 2.4: The Perceptron model (Minsky et. al. 1969)

As described in Sarker [21], linear Neural Networks are characterized by several different factors. They consist of multiple layers of neurons where the input layer receives an input and transforms it by propagating it through the network until the output layer returns the model's prediction. Between the input and the output layer exist multiple hidden layers which define the depth of the network. These hidden layers use nonlinear activation functions which are illustrated in Figure 2.5, such as ReLU (Rectified Linear Unit), Sigmoid or Tanh. They measure the activation of each neuron during the forward pass and are used during the training of the model so it can learn and model the complex nonlinear relationships in the data.

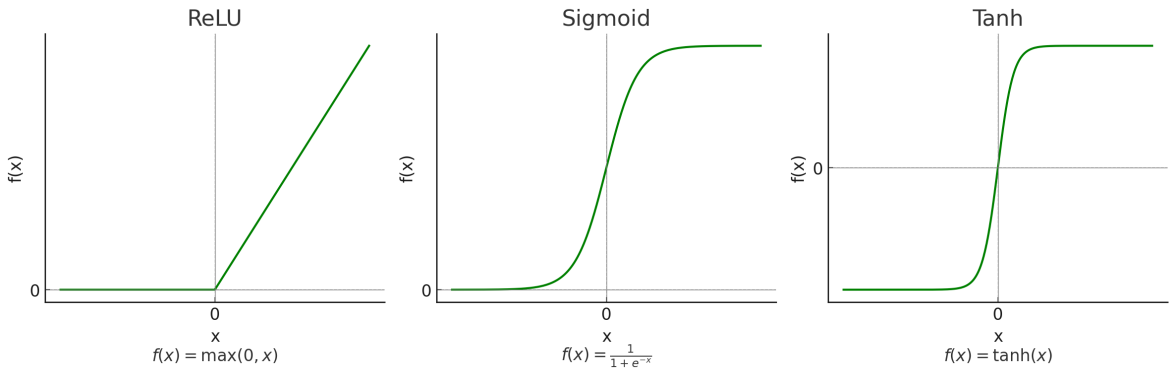

Figure 2.5: Activation Functions [22]

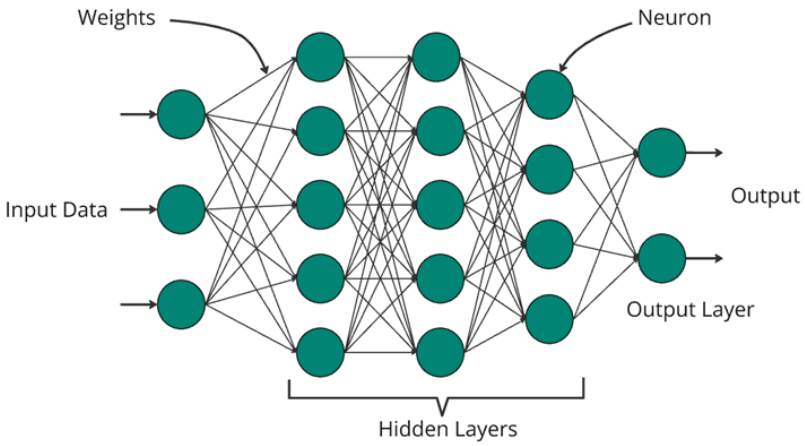

Figure 2.6: DNN [23]

The learning process of a network is an iterative process, where labeled training data is propagated from the input layer to the output layer [21]. The produced outcome can then easily be compared to the actual label, resulting in an error measure. The goal of training a model is to reduce this error by backpropagating it through the network and adjusting the network weights to improve the model's accuracy in making predictions. Various approaches were published for this process. They all have in common the use of the gradient of the loss function to change the weights, as it indicates the direction and rate at which each weight should change to reduce the error [21]. By calculating the gradient, we can determine the most effective direction to adjust the weights to minimize the loss function. During the backpropagation process, gradients are computed layer by layer, starting from the output layer and moving backward through the network to the input layer. This systematic process ensures each weight is updated based on its contribution to the total error. The process of adjusting weights is repeated over many iterations, gradually reducing the error and improving the model's performance. Figure 2.7 shows the intuition behind gradient descent (GD).

To update the model weights efficiently, gradient descent is commonly used. In traditional gradient descent, the gradient of the loss function is computed over the entire dataset, and the model weights are updated based on the average gradient from all data points [24]. This leads to stable and accurate weight updates, however, computing the gradient over the entire dataset in each iteration is highly computationally expensive as each update requires processing every training sample before adjusting the weights.

To address this challenge, stochastic gradient descent (SGD) was introduced [25]. Unlike GD, SGD updates the model after processing only a single data point or a batch of data points. This significantly reduces the computational time and leads to faster updates. The "stochastic" part comes from the randomness introduced by selecting a random sample or small batch for each update, rather than using

the full dataset. Although the updates in SGD are noisier and less stable than in traditional gradient descent, this randomness can help the model escape local minima and converge faster.

Backpropagation and gradient descent often get confused, which is why it is important to emphasize that Backpropagation is the process of calculating the gradients of the loss function with respect to the network's weights. Gradient descent, on the other hand, is the optimization algorithm that uses these calculated gradients to update the weights in order to minimize the loss function [26].

A stochastic gradient step can be mathematically described as:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W_{\text{old}}}$$

where, η is the learning rate, and $\frac{\partial L}{\partial W_{\text{old}}}$ is the gradient of the loss function L with respect to the weights W. To compute the optimal gradient descend in each step, optimizers like Adam or AdaGrad were developed. They work as mathematical heuristics to approximate optimal gradient steps for efficient and robust training [27].

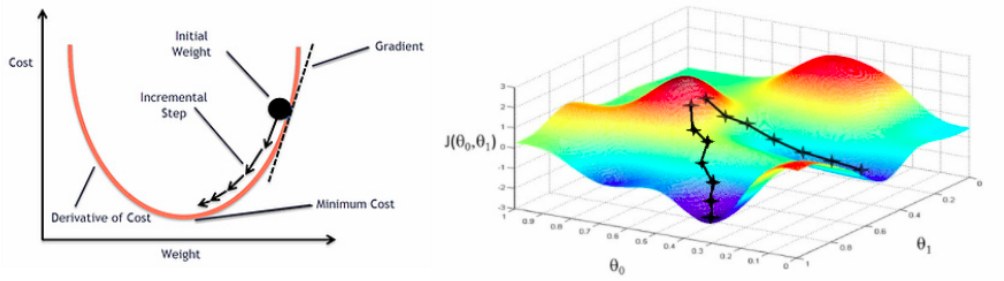

Figure 2.7: Gradient intuition [28] [29]

Unfortunately, these fully connected linear networks have drawbacks when applied to images. First, the dimensionality in images is significantly larger. For instance, a 256x256 color image has 196,608 features, as it has 256^2 pixels times 3 color channels. For fully connected DNNs, this ends up in a massive number of parameters, leading to increased computational load and memory requirements. Secondly, DNNs are not able to comprehend the spatial structure in images. They treat every single input feature independently, which means that they are not capable of detecting the relationship between adjacent pixels [30].

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were developed to address these limitations. They are specifically designed for processing grid-like data such as images and can gain knowledge from the spatial structure in images. CNNs are able to do so thanks to two additional features, convolutional layers and pooling layers. These are put in front of the fully connected layers from a standard DNN and enable the model to detect patterns and identify and enhance important features by down and upsampling the dimension in the image [31]. Figure 2.8 shows the basic structure of a CNN with two convolution layers as well as two pooling layers and the fully connected layer at the end.

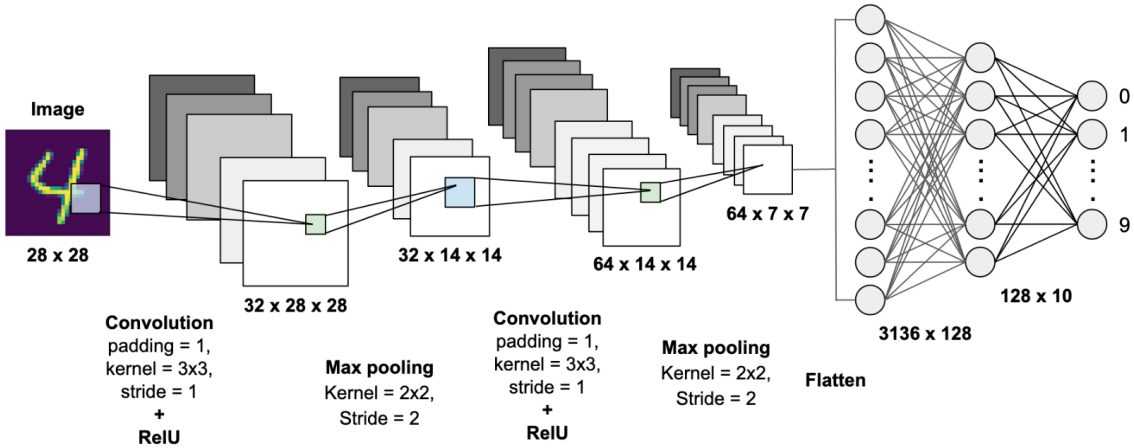

Figure 2.8: Convolution neural network [32]

2.1.3 Convolutional Layers

Convolutional Layers are often considered the core building blocks of a Convolutional Neural Network (CNN) [33] [34]. They consist of a set of filters, also known as kernels, which are matrices with $n \times n$ parameters. These weights are learned during the training process. Each convolutional layer contains several filters, and their number typically increases as the network depth grows.

Each filter is systematically moved, or "slid," across the image to compute a feature map that highlights specific features in the input data, as shown in Figure 2.9. It detects patterns within the image, such as edges, textures, or shapes where each filter is unique and responsible for identifying different patterns in the data.

Initially, the weights in the $n \times n$ filters are assigned randomly but are progressively refined during training. The training process of CNNs follows the same iterative approach as fully connected networks, employing backpropagation and gradient descent to minimize the error by adjusting the filter weights [35]. Convolutional layers are particularly efficient because they focus on the spatial structure of the

data, capturing relationships between nearby pixels. Unlike fully connected layers, which have many parameters due to dense connections, convolutional layers require fewer parameters and are optimized for detecting spatial patterns. This makes them especially effective for tasks involving image data [36].

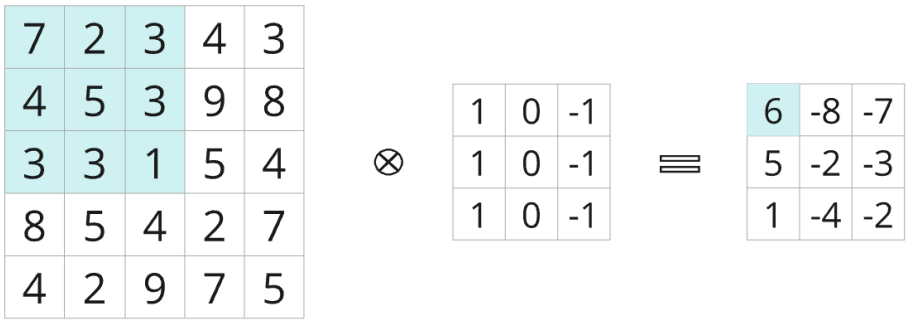

Figure 2.9: Filter

2.1.4 Pooling Layers

Pooling Layers are placed after the convolutional layers. They are also known as down-sampling layers and provide an approach to reduce the dimensions of the feature maps, which were increased by the convolution layer [36]. The pooling operation involves sliding a two-dimensional filter over each channel of the feature map and summarizing the features lying within the region covered by the filter. The two most common pooling methods are average pooling and max pooling, which summarize the average presence of a feature and the most activated presence of a feature respectively [35].

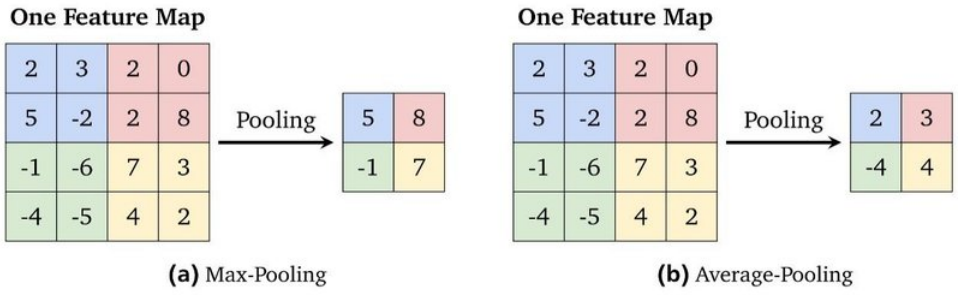

Figure 2.10: Max Pooling and Average Pooling [37]

Finally, after transforming the data through several convolutional and pooling layers, several fully connected layers are added after to perform high-level reasoning.

They aim to turn the insights about the spatial structure of the images into global semantic information, eventually concluding into a classification probability [38]. This conclusion is made at the last layer of the CNN, the output layer. For our classification problem in particular, it is common to use the softmax operator, which is shown below.

$$\sigma(z) = \frac{\exp(z_i)}{\sum_{j=1}^{D} \exp(z_j)}$$

It converts a vector of raw prediction scores into probabilities to represent a distributing probability for each existing class [39]. This simply means that the model output values are scaled to represent relative "probabilities" that a certain category was predicted by the network [40].

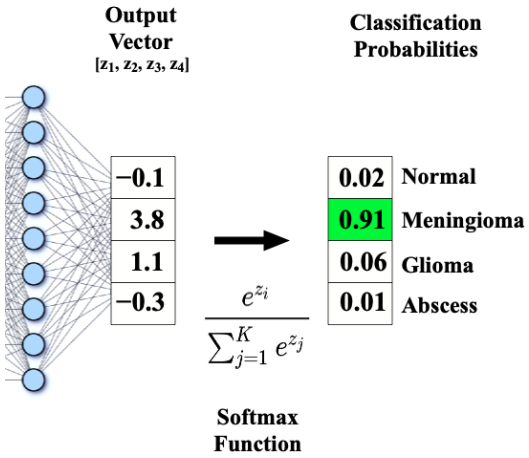

Figure 2.11: Softmax Operator [40]

2.1.5 Image Definition

Lastly, it has to be clarified how images are defined. Obviously, everyone knows what images are. However, computers do not perceive them as humans do, which is why a mathematical representation is necessary to represent images in a computer. As described in Mohan et al. [41], an image is a 2D grid of discrete points or pixels. Each pixel is defined by its image coordinates and color, with the origin of the coordinates located in the top left corner of the image. The color of an image can be represented in various ways: In grayscale images such as in Figure 2.12, where each pixel has a brightness value between 0 and 255 stored with 0 being completely black and 255 being completely white. In color images, separate values for red (R), green (G), and blue (B) are stored for each pixel, typically using 3 bytes for each color per pixel. Each color component can have a value between 0 and 255, allowing for the representation of a wide range of colors through their combination.

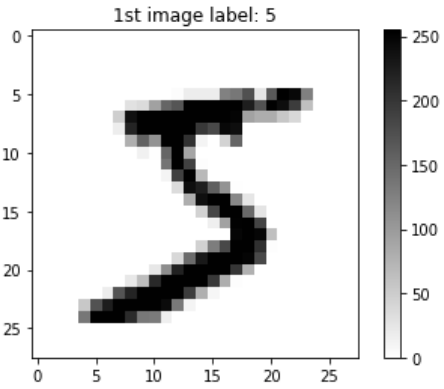

Figure 2.12: Representing a grayscale image from MNIST Dataset [42]

So far we have explored the fundamentals of Deep Learning in Computer Vision and its application for classification. We gained an understanding of the structural design of convolutional networks, including different kinds of layers, the activation functions, input type and size, pooling and convolution operations, classifier mechanisms and the training of these networks. This introduction only covers the basics and is essential as it moves on to the next section on Explainable AI.

2.2 Explainable AI

2.2.1 The role and importance of explainable AI

As AI systems continue to advance, they are taking over more and more complex tasks and they even outperform humans in areas such as driving, coding, or creative writing. However, what differentiates a human decision from an AI decision is that a human can argue their decision with reason, while Artificial Intelligence produce output based on complex mathematical models. This makes the decision-making process difficult to understand. Therefore, a severe interest has risen to know the underlying workings of Artificial Intelligence, opening the area of explainable AI.

XAI is a broad and versatile concept that refers to techniques and methods used to make the decision-making processes of AI systems transparent and understandable to humans. By explaining the decision of a model, we can foster **trust** into AI systems and overcome the "black box" character of these models. Additionally, this gained **transparency** enables **verification** of model decisions. For example, if an autonomous car suddenly changes its routes or a medical diagnosis system identifies a disease, explainable AI can verify that the sudden change in the route was caused by a traffic jam and the diagnosis was based on certain symptoms. XAI can also ensure **fairness** [43] and **accountability** [44] and prevent discriminatory outcomes such as biased loan rejections in credit lending systems.

From another perspective, explainable AI can be used to find **causality** by detecting undesirable correlations or patterns learned during training [45]. An interesting example is provided by Riberir et al. [46], as illustrated in Figure 2.13 where he demonstrated how a CNN trained to classify wolves and huskies relied on snow in the background instead of the animals themselves. This highlights how models often rely on unintended patterns, which XAI can help identify and underscores the importance of XAI in ensuring models make logical, fact-based decisions.

In summary, the key purposes of XAI are to build trust, verify decisions, address ethical concerns, and identify causality in AI systems [9]. This thesis, however, will demonstrate that XAI can also be leveraged to enhance the efficiency of model training and by this broaden its field of application.

(a) Husky classified as wolf (b) Explanation

Figure 2.13: Husky image and it's explanation [46]

2.2.2 Interpretability vs Explanation

The terms "explainability" and "interpretability" are often used interchangeably in discussions about XAI but represent distinct concepts. According to Barredo Arrieta et al. (2019), interpretability refers to how easily a human can understand the explanation provided by a model, while explainability describes the ability or any method that a model uses to clarify or reveal its internal workings or decisions [9]. So interpretability evaluates how comprehensible the explanation is to a human observer, whereas explainability focuses on the process of generating the explanation itself [3]. Much of XAI research focuses on improving the information gain from explanations and ensuring they are easily interpretable for humans. However, this thesis takes a different perspective. Here, the primary goal is to repurpose these explanations not just for better understanding but to enhance model performance.

So far we discussed the perspective on explainability, and we will now examine the different concepts of XAI. This will help us to characterize existing approaches and provide us with an extensive view of the whole field of XAI. Figure 2.14 highlights the concepts of interest, and we can use its sections from 1 to 5 like a red thread, guiding us through each level. This overview is based on the summaries provided by Belle and Papantonis [47] and Arrieta et al. [9].

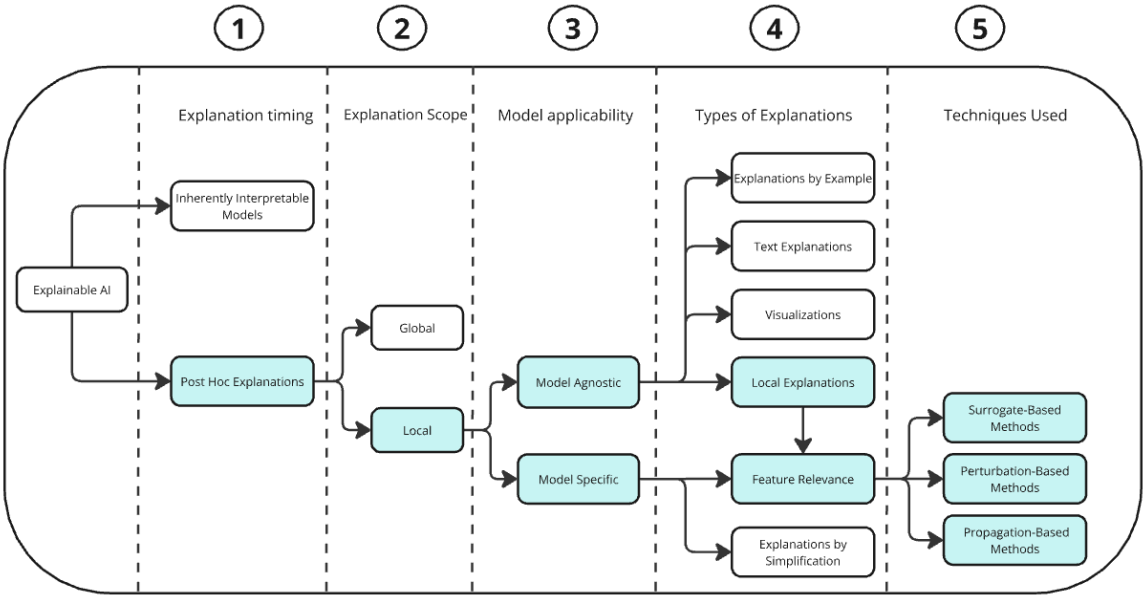

Figure 2.14: XAI Overview, blue nodes mark the points of interest of this work

2.2.3 Ante-Hoc and Post-Hoc Explanations

For simple machine learning models, the best explanatory models are themselves, as pointed out by Lundberg and Lee [48]. For example, the decision made by a decision tree is easily understandable by tracing down the nodes and edges of the specific tree leading down to its final decision, as visualized in Figure 2.15. For a trained linear regression model where the prediction y is defined by

$$y = \beta_0 + \beta_1 x + \beta_k x_k + \epsilon$$

[14], the decision can be understood by looking closely at the trained β's values. Let x_1, x_2, x_3 be the input we want to predict, then the values of $\beta_1, \beta_2, \beta_3$ indicate which variable attributes to the outcome value and which is not influencing it at all. These models are referred to as "white-box" or “glass-model” as analyzing the parameters of the model or its internal decision process without having the need for an external model is enough to provide an explanation [15].

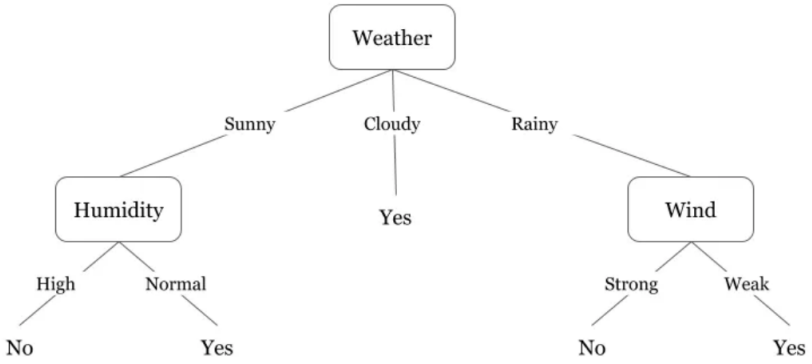

Figure 2.15: Decision Tree classifying if it would rain or not [49]

Such transparent, inherently interpretable models, also called ante-hoc models [50] are located in the realm of traditional machine learning and do not apply to network-based Models. The most eminent methods are linear andlogistic regression, decision trees, K-nearest Neighbours, rule-base learners, general additive models and Bayesian Models[51].

All of them have in common that their functionality and the way they operate are inherently interpretable for humans, allowing decisions to be traced directly through their processes. On the other hand, this level of interpretability does not apply to deep neural networks. As discussed in the introduction earlier, DNNs are highly complex, with an over-parameterized and high-dimensional nature that makes their decision-making processes incomprehensible. Therefore, these models cannot be explained during their decision-making runtime which makes it necessary to generate an explanation after the classification. This is referred to as post-hoc explainability [52].

All of the methods discussed in Chapter 3 are categorized to post-hoc models, which is why the next key points are subordinate to post-hoc explainability.

2.2.4 Global and Local explanations

The scope of an explanation refers to how specific or how general an explanation can be and is another important concept in XAI. We differentiate between two categories: **local** and **global** explanations.

Global explainability aims at understanding the overall decision logic of an ML model while local interpretability aims at understanding specific decisions for specific instances made by that model [53].
All of the methods discussed in this work are local methods. They were selected because we are interested in finding ways how a model can predict single samples better by leveraging XAI. Local explainability methods attempt to explain what features of the input may have contributed to the model's output.
Common global methods include filter visualization [39], activation maximization [40], partial dependence plots (PDP) [41], and confusion matrix analysis [42]. Interestingly, Feature Importance [43], often seen as a local method, can also provide a global perspective by generalizing feature attributions across the dataset. However, since this thesis does not focus on global model behavior, such methods are beyond its scope.

2.2.5 Model Agnostic and Model Specific Techniques

Some of the existing XAI methods rely on the inner structure of the network and need direct access to it. The nodes and edges contain valuable information which can be utilized for generating an explanation. On the contrary, some methods do not need any detail of the model and only focus on the input and its output. Their idea is to analyze the effect on the output when changing input. We therefore distinguish XAI methods on two properties: model-specific and model-agnostic [54].

Model-specific techniques are tailored to the unique characteristics and inner workings of a particular model and often use reverse engineering to analyze the internal decision-making process, which allows detailed and accurate explanations. However, this dependency can lead to consistency issues when comparing explanations across different models [54]. .

Model-agnostic techniques, on the other hand, work externally by analyzing the input-output relationship without requiring access to the model's inner workings. These methods are versatile and can be applied to various models [55], making them especially valuable in cases where access to a model's internal structure is restricted due to proprietary constraints or compliance issues [56]. Nevertheless, such restrictions are rarely an issue in research contexts.

Both approaches have their strengths and weaknesses which makes the decision for a certain approach highly dependent on the the specific goals, constraints, and conditions of the task at hand [57].

2.2.6 Explanation and its types

Clarifying how the explanation should be defined is the first step in developing explainable models [15]. In the context of XAI for classification in computer vision, explanations reveal why a model classifies an input image into a certain class. As explained earlier, the network structure is opaque, and analyzing the activations in each neuron and knowing which neuron is activated and which is not would not give a direct understandable clue as to why a predictor made its classification. Instead, it has turned out to be a proven method to put the focus on the input in order to create a comprehensible explanation.
When we see an image of a cat, we directly know it is a cat. But why? It is because the image has several features, such as whiskers, pointy ears, sharp teeth and fur, which combined make him classify it as a cat. The reasoning of a CNN model is based on a similar approach, extracting certain features and analyzing them together to classify them based on their previous training knowledge. So in order to understand why a model classifies the same image correctly or even incorrectly, it is necessary to analyze which features impact the decision positively and negatively so it is possible to find out which parts of the image were important to the prediction, and which did not. This is called feature attribution.

Formally, feature attribution methods aim to explain the prediction of the model for a specific input X by determining how much the i_{th} input feature x_i, for example, a pixel in an input image, has contributed to the classification decision [58]. To provide a complete understanding of the model behavior, it is necessary to analyze both positive and negative attributions to the model's output prediction. By this, we can for example modify the model to put higher focus on regions with high positive attributions to improve model accuracy.

Alternatives to Feature Attribution
Feature attribution methods are not the only variant for generating explanations, yet they are the most widely explored and popular approach [48][59][47]. According to Belle and Papantonis [47], post-hoc explainability can be differentiated into several other explanation methods next to feature attribution methods, including explanations by example, text explanations, visualizations, local explanations and explanations by simplification. These alternatives may be valuable for different applications, however, not all of them are suitable for the aim of efficient model training, **Explanations by example** for instance aim to justify the model predictions by showing similar instances from the training data that led to similar outcomes [60]. Although it is intuitive for humans, this method is impractical for guiding adjustments to the training process itself. **Text explanations** aim to provide human-understandable descriptions of the model's decision-making process but generating high-quality text explanations often requires substantial manual effort and domain-specific customization, which makes the method unsuitable for valuable integration into model training. **Visual explanation techniques**, such

as sensitivity analysis (SA) and individual conditional expectation (ICE) plots [61], provide graphical representations of how features influence predictions but focus on global behavior rather than specific samples and are therefore impractical for improving model accuracy [47].

In contrast to these alternatives, **local explanations** and **explanations by simplification** are often tied to feature relevance, as many of their methods identify influential features. This connection to feature relevance makes them particularly important for our work. Local explanations interpret model behavior around specific instances, which approximate decision boundaries by perturbing inputs and analyzing the impact on outputs. Explanations by
simplification replace complex models with interpretable surrogates, such as decision trees or rule-based systems that mimic the original model's behavior.

The declared aim was to utilize XAI for more efficient model training. Therefore, feature relevance is the most suitable approach as it can show the most influential features inside a sample, which could be used to nudge the model into focusing more on these features. By this, we could fit the model to focus more on specific features which possibly could lead to higher accuracy.

2.2.7 Surrogate-, Perturbation and Propagation-based Models

So far we have been exploring the most significant aspects of classifying XAI Methods. The first one is the explanation timing which distinguishes between post-hoc and ante-hoc methods, then the explanation scopes we specified reaching from local to global explanations, after which the model applicability were emphasized including the model agnostic and model-specific models. Lastly, the different types of explanations were described.

These aspects were all only describing the conceptual characteristics of XAI methods. The focus now shifts to the techniques used in feature attribution. Like in Samek's book Explainable AI: Interpreting, Explaining and Visualising Deep Leerning" [62], we will categorize the XAI techniques into three classes: surrogate-based, perturbation-based, and propagation-based explanation methods.

Surrogate-based Models
As explained earlier, simpler models such as linear regression or decision trees are inherently interpretable, while network-based models, with their multiple layers of non-linear transformations, are much harder to explain. Surrogate-based models address this challenge by approximating complex models with simpler, linear functions to generate explanations for their decisions [62]. The most well-known method in this category is LIME [46], which will be introduced later.

Propagation-based Models
Assuming we can access the parameters of the classification model, propagation-based methods use information such as gradient and activation, or even dedicated attribution values stored in each layer and neuron [62]. It performs a forward pass to predict the class label and then a specially designed backward pass to propagate the attribution back through the network to the input layer to estimate the feature attributions. The strategy for the backward pass differs in rules and processes depending on the method. What distinguishes propagation-based methods from the other methods is that the backward pass can be computed with the time complexity of the standard backward pass. In contrast, perturbation and surrogate-based methods often involve repeated sampling or perturbations, which can result in high computational costs [61].

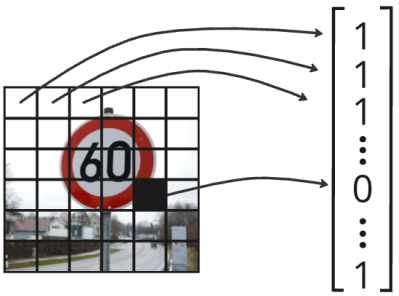

Figure 2.16: 6x6 pixel image with $\mathbb{R}^{36}$ interpretable representation [63]

Perturbation-based Models
Instead of focusing on the model's structure or process, we can also try to gain insights from the inputs and outputs of the model. Analyzing how a model's output responds to changing the model's input provides hints about which features in the input correlate with the model's output, thereby identifying which features cause the model to make a certain decision. The scale of perturbation depends on the method, and it can be exponentially high as we have nxn pixels. This is visualized in Figure 2.16 To be able to measure the effect of a perturbation, two essential tools are used: **baselines** and **distance metrics** [64].

Baselines are used as a reference point to understand the effect of perturbations on the input image. This approach helps in identifying which parts of the input image are most important for the model's predictions by comparing the perturbed image to the baseline. The original image could be the baseline, or an image where every pixel is set to zero, ergo a black image or so-called "Zero Baseline". It is also possible to create several baselines and compare them to the perturbation.

A distance measure is crucial to accurately assess and scale the impact of the input modifications we create in the perturbations. It quantifies the difference between the model's predictions before and after the perturbation [46]. By doing this, we can compare perturbations and rank the importance of various regions or features. Commonly used distances are the euclidian, manhatten or cosinus distance. If a

small perturbation, which means a close distance, causes a significant change in the model's output, it indicates high sensitivity. This part of the image is therefore crucial for the model's prediction. Small changes require more perturbations to be able to get a meaningful result, leading to computational overhead.

It is possible to combine the strategies of surrogate, propagation, and perturbation-based models. For instance, one can first simplify the model or the data sample using simplified representations and then apply perturbations to these simplified versions. An example for this is again LIME [46]. Another option is to combine surrogate models with propagation-based methods. DeepLIFT (Deep Learning Important FeaTures) [65], for example, compares perturbed inputs with a reference baseline and propagates the differences back through the network to identify the contributions of each input feature to the output.

The end result of these three strategies can be visualized in the same dimension as the original image, highlighting how each pixel contributes to the prediction on a scale as shown in Figure 2.17. Depending on the visualization method chosen and the implementation, this scale typically ranges from [-1, 1]. Negative numbers indicate a negative contribution to the outcome, while positive values show the pixels with a positive contribution.

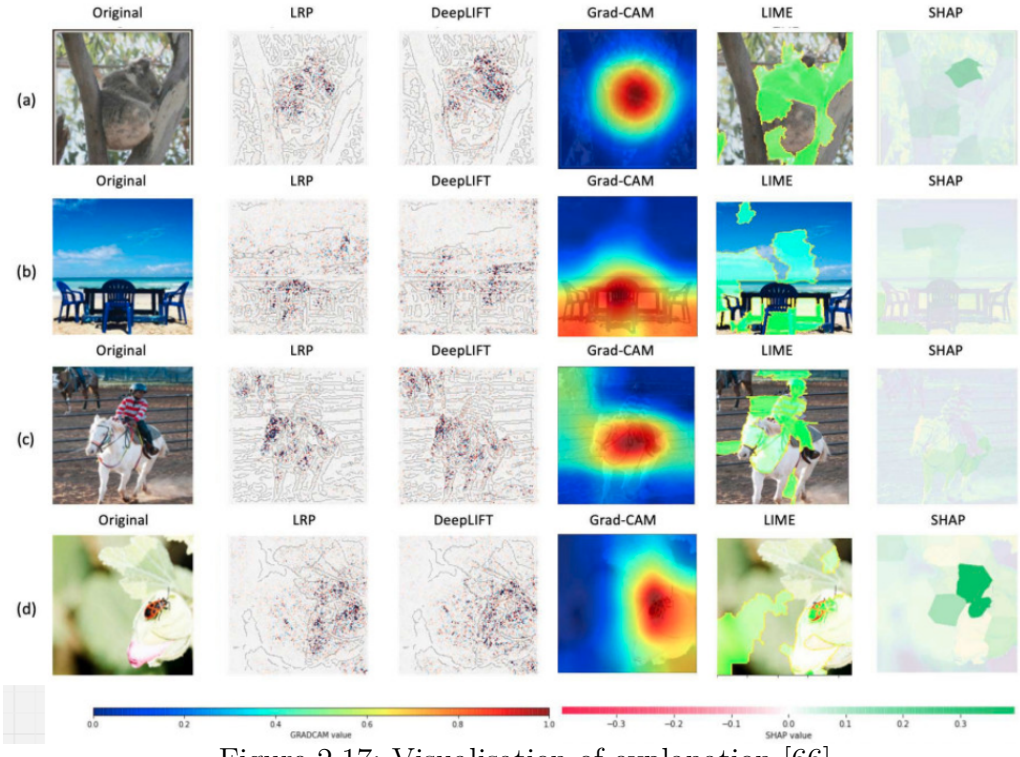

Figure 2.17: Visualisation of explanation [66]

3 xAI Methods

3.1 LIME

The first XAI method we discuss is LIME, which stands for Local Interpretable Model-agnostic Explanations and was presented by Ribeiro, Singh, and Guestrin [46]. The basic idea behind LIME is to explain model predictions by altering a simplified inputs and to analyze how the prediction changes in response to these perturbations to identify the attributing features. By perturbing the input, LIME approximates the model locally in the vicinity of the given sample and learns a sparse linear model around it that serves as a human interpretable explanation. LIME is therefore a surrogate and perturbation-based technique. To explain Lime more in detail we will use Figure 3.1 from Captum [67] as an intuitive guide.

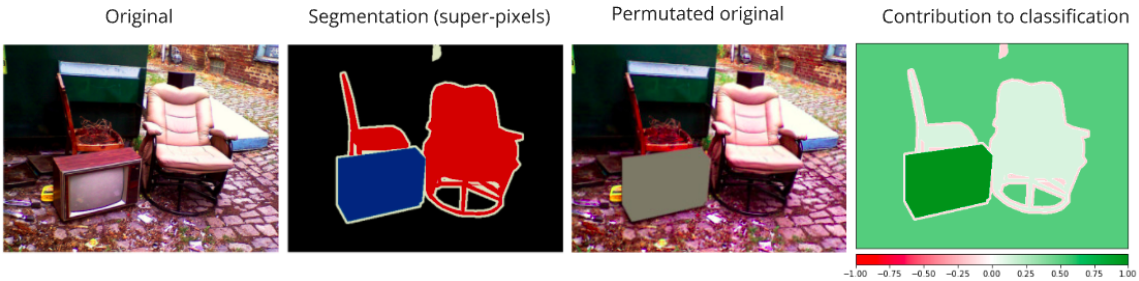

Figure 3.1: LIME Demonstration

This image illustrates the main steps of LIME as it generates an interpretable explanation for a model's prediction. It starts with the model classifying an image of furniture with "television" as the most probable label. The exact details of the model are unknown to LIME because LIME relies only on the input instance X and its predicted outcomes.

LIME simplifies the image by converting it into a binary vector $z \in \{0, 1\}$, indicating the "presence" or "absence" of each pixel in the image. These pixels can even be patched as superpixels to simplify and reduce the number of perturbations. For demonstration purposes, the example image is already segmented into perfect superpixels. This representation allows for a manageable number of combinations. In this case, four segments lead to $2^4 = 16$ possible perturbations. However, already pre-segmented images are rarely the case for real-world data.

The model then re-evaluates each of these perturbed samples, indicating how the removal of specific segments affects the likelihood of a particular classification, such as "television." By examining how these changes impact predictions, LIME identifies which segments are most influential in the model's decision. To prioritize meaningful samples, LIME uses a distance measure like Euclidean distance to assign higher weights to perturbations similar to the original image.

Finally, LIME fits a simple linear model, such as Lasso regression, to the weighted samples to approximate the influence of each segment on the classifier's prediction. The final result highlights the most influential segments, as shown in the last panel of Figure 3.1.

Through this process, LIME offers a localized, instance-specific explanation by simplifying the complex model's behavior around a specific instance X. By converting the image into a binary vector of segment presence and absence, LIME provides a human-understandable insight into the model's predictions without needing access to its internal workings.

3.2 SHAP

3.2.1 Shapley Values

To understand SHAP, it is necessary to be familiar with the concept of Shapley values. Shapley values originate from cooperative game theory and were introduced by Lloyd Shapley in 1953. This technique fairly distributes the total payoff from a cooperative game to the game's players [68].

The Shapley value for a feature quantifies its contribution to the model's prediction, calculated by averaging its marginal impact across all possible subsets of features. Given a set of players N and a value function v that assigns a value to every subset $S \subseteq N$, the Shapley value ϕ (v) for a player i is calculated as:

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left(v(S \cup \{i\}) - v(S)\right)$$

This formula ensures that the contribution of each player i is averaged over all possible coalitions S that do not include i. In the context of image classification, the value function v is used to represent the output of the model which is the predicted probability for a given subset of features. The value function $v(S)$ gives the model's prediction for the subest S of features excluding feature i, while $(v(S \cup \{i\}) - v(S))$ represents the change in the model's prediction when the feature i (a pixel or superpixel) is added to the subset S.

3.2.2 SHAP (SHapley Additive exPlanations)

SHAP is a unified framework that applies the concept of Shapley values from cooperative game theory to explain the predictions of machine learning models. When applying the Shapley values method to machine learning models, the key step is to set up a cooperative game, in which players are the input features and their payoff is the model prediction [69]. In our case of image classification, the image's features are the players while the payoff is the classification.

For each feature i in X, SHAP calculates how the prediction changes when i is added to every possible subset S of the other features:

$$v(S \cup \{i\}) - v(S) \tag{3.1}$$

Here, $v(S \cup \{i\})$ is the model's prediction with feature i included in subset S, and $v(S)$ is the prediction without feature i.

The SHAP values are obtained by averaging these marginal contributions, weighted by the number of permutations of the feature set:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left(v(S \cup \{i\}) - v(S)\right) \tag{3.2}$$

This ensures that the contributions are fairly distributed among all features.

Computing Shapley values involves evaluating the game payoff for every possible combination of features. This makes the computation exponential in the number of features. For games with few players, it is possible to exactly compute the Shapley values, but for games with many players, the Shapley values can only be approximated. Several approximation methods have been introduced to address this issue. The methods that are most suitable for image classification are **Kernel-SHAP** and **DeepSHAP**. These are specific implementations or approximations within the SHAP framework. Kernel SHAP is a model-agnostic and concrete implementation of the SHAP framework we explained earlier, using a weighted linear model to approximate the attribution of each feature [48]. In contrast, Deep SHAP combines the backpropagation technique of DeepLIFT, a method that we will discuss later in this thesis, and leverages the structure of neural networks to compute feature attributions more efficiently [48].

3.3 Propagation based methods

As introduced earlier, the challenge of perturbation-based methods used to provide explanations in image classification lies in the exponential increase in computational complexity due to the exponential increase in the number of necessary perturbations.

In contrast, propagation-based approaches offer a more efficient method. These approaches propagate an importance signal from an output neuron backward through the network layers to the input layer in a single pass, significantly reducing computational overhead [70]. Some of these methods utilize the gradient of the model's output with respect to the input, which is why these methods are also referred to as gradient-based methods [71]. Mathematically, the gradient of the model is defined as $\frac{\partial f}{\partial \mathbf{x}}$. It includes every node in the network and reflects how changes in input features affect the model's output. Methods such as Guide Backpropagation, SmoothGrad Gradient * Input, CAM, and Grad CAM can efficiently compute feature importance without the need for extensive perturbations. These methods have been developed progressively, each attempting to overcome the limitations of its predecessors. Alternatively to computing the gradient, DeepLIFT [70], and LRP [72] have their own methods of propagating the attribution back to the input layer. But all of these approaches have in common to leverage the network's internal structure to trace the contribution of each input feature to the final prediction. We will introduce them one by one, especially focusing on the most promising methods, Grad Cam, DeepLift, and LRP.

3.3.1 Saliency Maps

Saliency maps, which were introduced by Simonyan, Vedaldi, and Zisserman [73] are among the simplest propagation techniques. In these methods, the backpropagation process calculates the gradient of the output with respect to each input neuron, effectively showing the importance or "saliency" of each neuron relative to the model's prediction. This information can be visualized using a saliency map. Saliency maps often use the absolute values of the gradients. This means they highlight regions where the model is most sensitive, but they do not distinguish between positive and negative contributions. Therefore, saliency maps highlight where in the image the model is focusing on, thus providing insights about the decision boundary rather than showing which pixels contribute positively or negatively to the decision. However, saliency maps can be modified to show both positive and negative attributions by visualizing the raw gradients instead of their absolute values. Although saliency map offer an uncomplicated approach for generating an explanation, its explanation can be noisy and they struggle with vanishing gradients which limits interpretability in deeper networks.

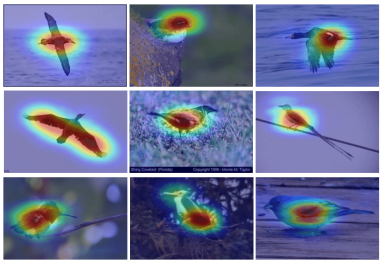

Figure 3.2: Saliency Maps [74]

3.3.2 Deconvolutional Network

Closely related to Saliency Maps are Deconvolutional Networks which were introduced by Zeiler and Fergus [75]. They aim to visualize the activations within a CNN by reversing the convolution operations. This process works by passing an activation map backward through the network using transposed convolution, which is also referred to as deconvolution. Unlike saliency maps, DeconvNets suppress all negative gradients during backpropagation. This results in a visualization that highlights only positively contributing features what simplifies the interpretation but also misses negatively contributing factors, which can be critical for understanding model decisions.

The most conceptual difference between Saliency Maps and DeconvNets lies in how they handle negative gradients during backpropagation, particularly at ReLUs [50]. In Saliency Maps, negative gradients are allowed to pass through the ReLU by which both positive and negative contributions can be retained. In contrast, DeconvNets set all negative gradients to zero at the ReLU, regardless of the input's behavior during forward propagation. As a result, DeconvNets focuses solely on positively contributing features and offer a simpler but less comprehensive visualization. Saliency Maps can capture both positive and negative contributions, visualizing their absolute values and providing a more balanced view of the input features. However, DeconvNets tend to highlight only positively contributing features and miss out on negatively contributing factors which are crucial for understanding and improving the model.

3.3.3 Guided Backpropagation

Guided Backpropagation by Springenberg et al. [76] combines the concepts of Saliency Maps and DeconvNets. Like Saliency Maps, it computes the gradient of the output with respect to the input features, but during backpropagation, it filters out negative gradients at ReLUs, as in DeconvNets. This ensures only positive gradients contribute to the visualization, resulting in clearer and more detailed maps of features that support the model's decision. However, as the two methods before,

Guided Backpropagation fails to highlight inputs that contribute negatively to the output.

3.3.4 SmoothGrad (SG)

SmoothGrad [77] was introduced to address the inherent noise and instability which are often found in gradient-based explanations. Traditional methods often create messy and hard-to-read attribution maps because they are too sensitive to small changes in the input. SmoothGrad reduces this noise by averaging the gradients of multiple perturbed versions of the input. This results in smoother and more visually interpretable maps. The gaussian distribution is used to generate these "noisy" copies of the input and the smoothed gradient is then computed as the mean of the gradients over these noisy inputs. This can be formally described as:

$$SmoothGrad(x) = \frac{1}{n}\sum_{i=1}^{n} \nabla_x f(x + \mathcal{N}(0, \sigma^2)) \tag{3.3}$$

3.3.5 Gradient * Input

The Gradient * Input method [78] multiplies the gradient with the input itself and was proposed as a technique to improve the sharpness of the attribution maps. Mathematically it can be described as

$$\text{Attribution} = x_i \times \frac{\partial f}{\partial x_i}$$

.

In image data, this means the color intensity of a pixel is multiplied by the gradient of the output with respect to that pixel. Unlike methods that use the absolute gradient, Gradient $\times$ Input preserves the sign of the gradient, allowing it to distinguish between features that positively or negatively influence the output.

3.3.6 Integrated Gradients (IG)

Integrated Gradients use an integral to calculate the importance of each input feature by looking at how the model's output changes as the input gradually transitions from a baseline to the actual input [50] [79]. Instead of calculating feature importance at a single point, Integrated Gradients measures the model's behavior along the entire path connecting the baseline to the input.

This process can be thought of as starting with a blank or neutral input and gradually transforming it by a scaling factor α which varies from 0 as the basline to 1 into

the actual input step by step. At each step, the model's sensitivity to changes in the input is calculated using the gradient of the output with respect to the input. Mathematically, Integrated Gradients are defined as:

$$\mathrm{IG}_i(x) = (x_i - x_i') \times \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha \tag{3.4}$$

where x is the input, x' is the baseline input, α the interpolation scalar and $\frac{\partial f(x)}{\partial x_i}$ is the gradient of the model's output with respect to the i-th input feature.

The integral aggregates these sensitivity measurements across all steps along the path, capturing how feature importance evolves as more of the input is introduced. This provides a smooth and comprehensive measure of each feature's contribution to the final prediction. This approach provides robust and interpretable attributions, addressing challenges like gradient saturation and offering a more reliable perspective on feature importance.

3.3.7 CAM and Grad-CAM

In Section 2.1.2, we explained the structure of CNNs. After several convolutional layers and pooling layers, the signals are connected to a fully connected network for higher decision-making. At this transition point, Class Activation Mapping (CAM) comes into play. After forward propagating through the network, CAM combines the detected features with the important weights from the first fully connected layer of the network to analyze feature importance [80].

For CAM to function properly, certain structural conditions are necessary: the convolutional network must end with a Global Average Pooling (GAP) layer, which averages each feature map produced by the last convolutional layer across all spatial dimensions. This results in a single value for each feature map which represents the presence of that feature across the entire image as shown in Figure 3.3. These values are then multiplied by the corresponding weights in the first fully connected layer, which determines their contribution to the prediction.

However, despite the usefulness of CAM, it requires a specific architecture. The network must end with a GAP layer directly connected to a fully connected layer. To overcome this limitation, Gradient-weighted Class Activation Mapping, also known as Grad-CAM, was developed. Grad-CAM is more versatile, as it generalizes the concept of CAM and can be applied to any CNN architecture

Grad-CAM calculates feature importance by using the gradients of the target class score with respect to the feature maps $A_k(x, y)$ in the last convolutional layer [81]. These gradients are averaged across spatial dimensions (x, y) to compute the importance weights α_k^c for each feature map. The weights α_k^c represent how much the k-th feature map contributes to predicting the target class c. The final class

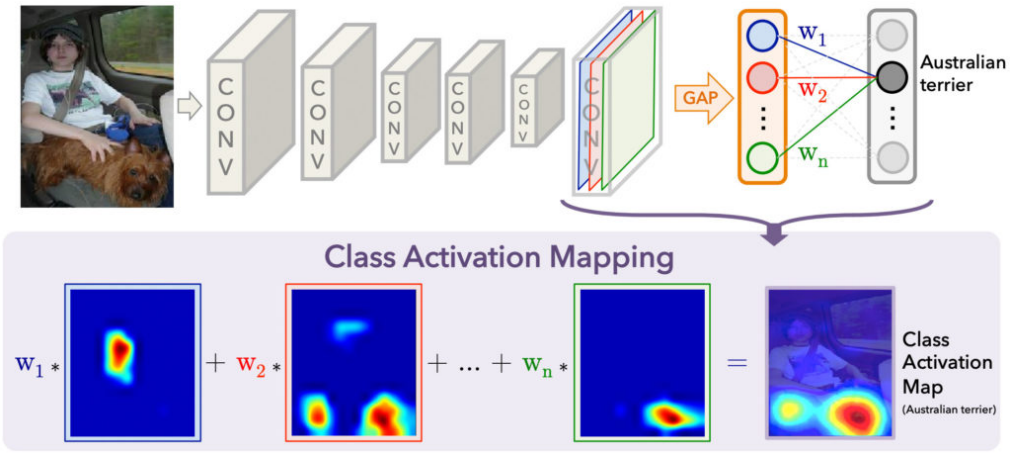

Figure 3.3: Class Activation Map [81]

activation heatmap is then computed by combining the feature maps $A_k(x, y)$ with their corresponding weights α_k^c, and summing the results.

The Grad-CAM heatmap is mathematically expressed as:

$$\text{Grad-CAM}_c(x, y) = \text{ReLU}\left(\sum_k \alpha_k^c \cdot A_k(x, y)\right)$$

In the original paper by Selvaraju et al. [80], A Relu is used at the end to ensure only positive attributions. This process produces a heatmap that highlights the regions most relevant for predicting the target class.

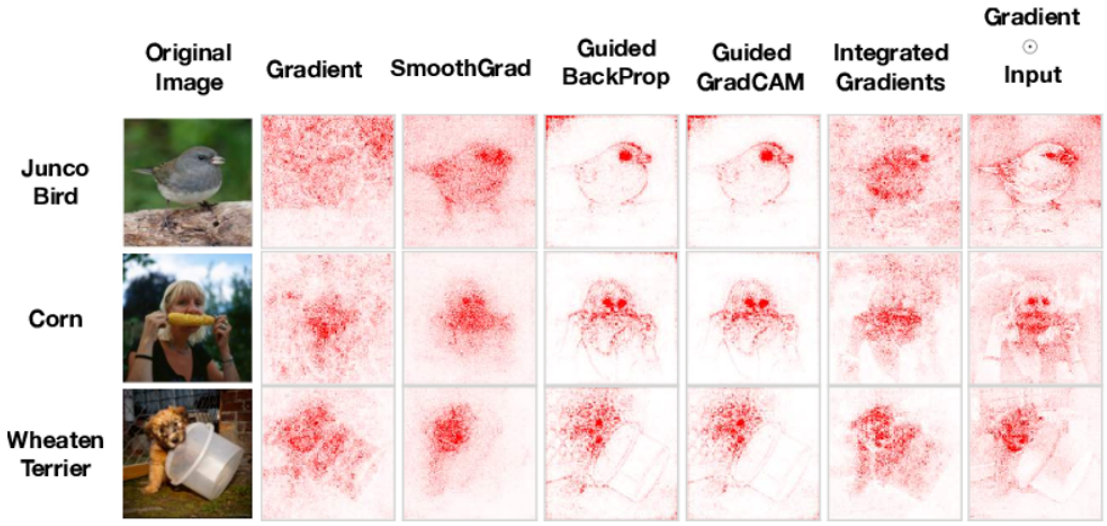

Figure 3.4: Comparing different propagation-based results

3.4 DeepLift

The main goal of DeepLIFT (Deep Learning Important Features) is to provide a more reliable way of attributing the importance of each input feature to the output of a model and to avoid common gradient-based issues such as vanishing gradients and saturation [72]. The core idea behind DeepLIFT is to compare the model's output for a given input with the output for a "reference" input which serves as a baseline. This comparison helps to understand how much each feature of the actual input contributes to the difference in the model's output.

The difference in the model's output between the actual input and the baseline is calculated as:

$$\Delta f = f(\mathbf{x}) - f(\mathbf{x}_0)$$

where x is the input image and x_0 is the baseline. This difference represents how much the model's output changes when moving from the baseline input to the actual input. DeepLIFT attributes the difference in the output Δf to each individual input feature by propagating this difference backward through the network. Instead of just looking at gradients, DeepLIFT then propagates these differences back through the network, using specific rules to determine how much each neuron in the network contributed to the change in output. These rules ensure that the contributions are fairly and accurately assigned. The choice of rule depends on the type of activation or interaction between neurons.

Linear Rule
For linear activations, meaning the output of the neuron is a direct, weighted sum of its inputs such as $z_i = \sum_j w_{ij} z_j$ without any additional non-linear transformation like a ReLU, the contribution is simply proportional to the input. For a neuron z_i in a layer i which receives inputs from neurons z_j in the previous layer, weighted by w_{ij}, the contribution from z_j to z_i is then:

$$C_{\Delta z_j \to \Delta z_i} = w_{ij} \cdot \Delta z_j$$

Rescale Rule
The Rescale Rule is used when neurons have non-linear activations (e.g., ReLU, sigmoid). It rescales the contributions to ensure the conservation of the total difference across layers. If a neuron z_i is activated by inputs z_j from the previous layer, the contribution $C_{\Delta z_j \to \Delta z_i}$ of each input z_j to z_i is

$$C_{\Delta z_j \to \Delta z_i} = \Delta z_j \cdot \frac{\Delta z_i}{\sum_j \Delta z_j}$$

Reveal-Cancel Rule
The **Reveal-Cancel Rule** addresses interactions between input features that amplify or cancel each other's effects on the model's output. It decomposes the total change in

the output (Δy) into **positive contributions** (Δy^+) and **negative contributions** (Δy^-).

These contributions are propagated backward through the network, where they are distributed among neurons at each layer based on their role in amplifying or canceling the output change. At the input layer, the positive and negative contributions are **combined and summed** to compute the final attribution for each feature. This ensures that features contributing to increases and decreases in the output are fairly represented and captures the interactions between inputs.

3.5 DeepSHAP

An extension of DeepLIFT is DeepSHAP, which combines DeepLIFT with SHAP [82]. Unlike DeepLIFT, which depends on a single baseline, DeepSHAP uses SHAP's perturbation principle to create multiple baselines. It then calculates attribution values for each baseline and averages them. This approach reduces noise sensitivity and produces more precise results. However, the computational cost increases significantly due to the additional baselines.

The last method we discuss now, Layer-wise Relevance Propagation (LRP) [72], might be a better alternative, as it overcomes DeepLIFT's limitations which arise from reliance on baseline inputs and the relative nature of its attributions as well as DeepSHAP's high computational burden.

3.6 LRP Layerwise-Relevance-Propagation

LRP is an attribution method that propagates the prediction backward through the network by redistributing the prediction score layer by layer until it reaches the input. Each neuron is assigned a relevance score based on its contribution to the final prediction by applying specific propagation rules at each layer, similar to DeepLift. But unlike DeepLIFT, which compares the output relative to a baseline, LRP focuses on the absolut contribution of each input feature without needing a reference point, making the relevance scores more absolute and stable.

3.6.1 Core Concept of LRP and Relevance Conservation

In LRP, relevance is propagated backward through the network in a manner that preserves the overall attribution score across layers. This means that the relevance scores calculated at each layer should sum to the same value as the prediction score, ensuring consistent scaling throughout the model. This property, known as relevance

conservation, is fundamental to LRP, as it maintains the integrity of the prediction score and allows each layer to be analyzed independently. Formally this property can be described as:

$$f(x) = \cdots = \sum_{d \in \mathcal{I}_{l+1}} R_d^{(l+1)} = \sum_{d \in \mathcal{I}_l} R_d^{(l)} = \cdots = \sum_d R_d^{(1)}$$

where $f(x)$ is the model's output prediction, and $R_d^{(l)}$ represents the relevance assigned to neurons at each layer. This equation guarantees that the total relevance remains the same as it flows back to the input, preserving the score from layer to layer.

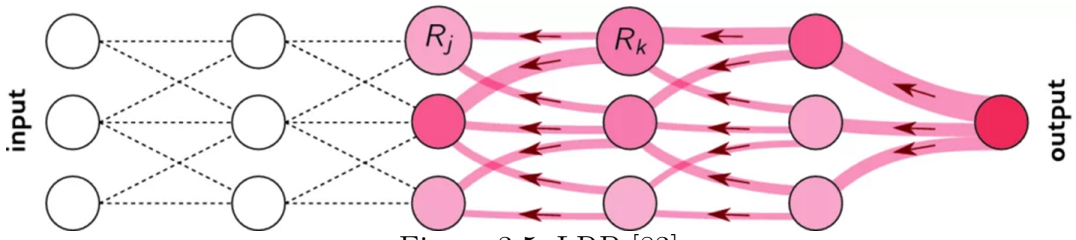

Figure 3.5: LRP [83]

3.6.2 LRP Propagation Rules

LRP assigns relevance scores at each layer using propagation rules, which determine how the relevance is distributed from the current layer to the previous one [83].

a. Basic Rule The most fundamental rule in LRP is the basic propagation rule [72], which ensures that the relevance of a neuron in the current layer is distributed among the neurons in the previous layer in proportion to their contributions to the activation of the current neuron. Mathematically, if R_i is the relevance score of neuron i in the current layer, and z_{ij} is the contribution of neuron j in the previous layer to neuron i, the relevance score R_j for neuron j in the previous layer is computed as:

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$$

b. LRP-ϵ Rule To handle numerical instabilities and avoid division by small numbers, the LRP-ϵ rule modifies the basic rule by adding a small term ϵ to the denominator [84] [83]:

$$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k$$

This ensures that the relevance propagation remains stable even when neuron activations are close to zero.

c. LRP-γ Rule The LRP-γ rule further modifies the propagation by multiplying the weights w_{ij} by a factor γ to increase the relevance of positive contributions, which can be particularly useful in scenarios where positive inputs are more significant than negative ones [83]:

$$R_j = \sum_k \frac{a_j \cdot (w_{jk} + \gamma w_{jk}^+)}{\sum_{0,j} a_j \cdot (w_{jk} + \gamma w_{jk}^+)} R_k$$

Here, w_{ij}^+ represents the positive part of the weight, ensuring that the relevance is more heavily assigned to positively contributing features.

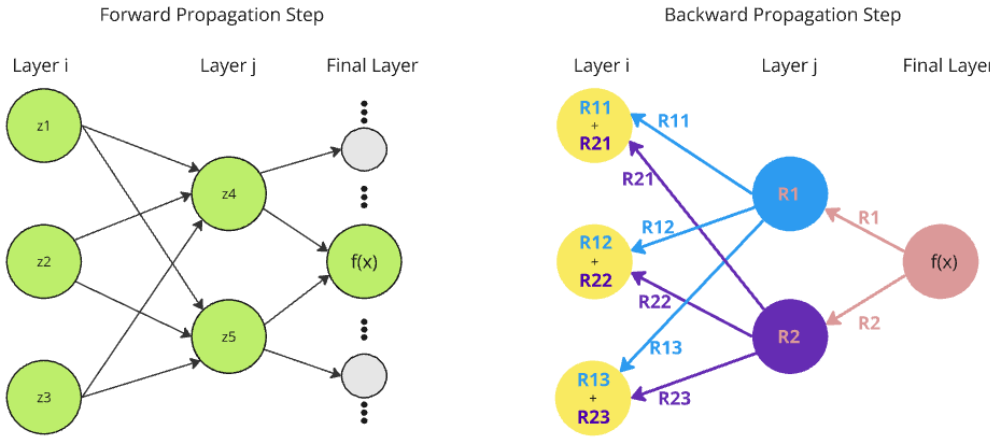

Figure 3.6: LRP Algortihm

3.6.3 LRP Algorithm

The LRP Algorithm is applied on a trained model and starts with a forward pass during which the activations and intermediate results in the individual layers are stored. Then starting from the output layer, the attribution is propagated backward through the network to the input layer. For each layer, the attribution is distributed to the neurons in the previous layer based on the previously discussed rules that depend on the connection weights and activations. This process is visualized in figure 6.3. After distributing the relevance through each layer, the backpropagation eventually end in the input layer, where the dimensions are exactly the same to the input layer with the difference that it does not contain color values but instead relevance values for each pixel.

In classification models, each neuron in the output layer represents a possible class the model can predict with a probability for its classification likelihood. Relevance propagation starts from this output layer, specifically from the neuron corresponding to the predicted class with the highest probability. The relevance score is initialized

using the prediction score $f(x)$ for this class. As shown in Figure 3.8, the relevance scores R_1 and R_2 for the output neurons are then calculated as:

$$R_1 = \frac{z_4 \times f(x)}{z_4 + z_5}, \quad R_2 = \frac{z_5 \times f(x)}{z_4 + z_5}$$

Once the relevance scores R_1 and R_2 have been assigned to neurons z_4 and z_5 in the intermediate layer, we propagate these relevance scores further back to the neurons in the preceding layer. This redistribution is done according to each neuron's weighted contribution to the activations of neurons z_4 and z_5.

For neurons z_1, z_2, and z_3 in the first layer, the relevance scores R_{11}, R_{12}, and R_{13} are computed by distributing R_1 from z_4 to each of these neurons based on their contribution. Similarly, relevance scores R_{21}, R_{22}, and R_{23} are computed by distributing R_2 from z_5 to each of these neurons:

$$R_{11} = \frac{z_1 \times R_1}{z_1 + z_2 + z_3}, \quad R_{12} = \frac{z_2 \times R_1}{z_1 + z_2 + z_3}, \quad R_{13} = \frac{z_3 \times R_1}{z_1 + z_2 + z_3}$$
$$R_{21} = \frac{z_1 \times R_2}{z_1 + z_2 + z_3}, \quad R_{22} = \frac{z_2 \times R_2}{z_1 + z_2 + z_3}, \quad R_{23} = \frac{z_3 \times R_2}{z_1 + z_2 + z_3}$$

In these equations, each relevance score R_{ij} represents the portion of relevance passed from neuron i in the current layer to neuron j in the previous layer. This redistribution ensures that the total relevance is preserved across layers, as each neuron's relevance in the current layer is proportionally allocated to the neurons in the preceding layer.

Finally, to obtain the total relevance for each neuron in the first layer (i.e., z_1, z_2, and z_3), we sum up the contributions from both R_1 and R_2:

$$R_{z_1} = R_{11} + R_{21}, \quad R_{z_2} = R_{12} + R_{22}, \quad R_{z_3} = R_{13} + R_{23}$$

3.6.4 LRP in different Layer Types

1. Fully Connected Layers In fully connected layers, where every neuron connects to all neurons in adjacent layers, relevance propagation follows the basic rule or one of its modified forms, distributing the relevance scores R_i proportionally to each connected neuron based on their weighted contributions z_{ji}.

2. Convolutional Layers In convolutional layers, relevance is propagated from each feature map in the current layer to its corresponding feature map in the previous layer, following the connections defined by the convolutional filters. Each filter determines the weighted contribution of neurons in a local region (receptive field) of the previous feature map to a neuron in the current feature map. During relevance propagation, the filter weights are used to redistribute relevance locally within the receptive field, ensuring that spatial structure is preserved.

3. Max Pooling Layers Max-pooling layers require a unique approach in LRP. Since max-pooling layers do not have learnable weights, relevance cannot be distributed proportionally based on weight contributions. Instead, LRP assigns all relevance to the neuron in the previous layer that had the maximum activation value, preserving the information about which features were deemed most significant during forward propagation.

4 Efficient Model Training and XAI

In the previous chapter, we discussed the state-of-the-art XAI methods, which are highly regarded in current XAI research. We examined their functionalities, strategies, advantages, and some of their limitations. Now we will define what we consider under efficient model training, how we want to achieve efficient model training, and then evaluate which of the presented method might be the most promising and effective method.

Efficient model training can take various forms [85]. Training is considered more efficient if it requires less time to complete which is often achieved through advanced optimizers that maximize the efficiency of stochastic gradient descent. Efficiency can also refer to faster convergence, where a model reaches the desired accuracy in fewer epochs. Additionally, a model is considered more efficient if it can maintain the same level of performance with fewer data. This is exactly the goal of this thesis, to enhance data efficiency by enabling models to achieve comparable accuracy with fewer training samples. As discussed in the introduction, data aggregation and data preprocessing are often time- and cost-intensive, and data availability is limited in many cases. By leveraging XAI, we aim to enable models to achieve high accuracy with fewer data. Additionally, this enhanced training approach fundamentally aims to improve accuracy, ensuring that models can achieve or even surpass previous accuracy levels. To evaluate the options available for achieving efficient model training, we first categorize the methods and then explore promising approaches.

In our analysis, it became clear that these XAI methods can be grouped into two categories: model-agnostic methods that rely on perturbations and sometimes surrogations, and methods that depend on the model's structure and are based on backpropagation or propagation of relevance scores. From these categories, we derive two fundamental approaches for integrating XAI into model training:

(1) Using XAI Externally for data sampling or sample weighting and

(2) Integrating XAI strategies directly into the model's inner structure

The following sections will explore these strategies in more detail to evaluate which method might be most effective for enhancing training efficiency.

4.1 Data Sampling

Data sampling is a commonly used technique in machine learning to improve training efficiency by selecting a representative subset of the dataset [86]. The primary benefit of sampling is that it reduces the overall dataset size which can potentially lead to faster training times without sacrificing the accuracy of the model. An XAI method can be used guide the sampling process by selecting or prioritizing samples based on their relevance to the decision-making of the model.

Sampling data starts with computing the relevancy score for each sample in the training data set. The next step is to select a subset of samples by sorting the samples based on a comparison metric and choosing the k-percent highest attributing samples. Here it is crucial to choose a good metric, which indicates the explanation value of a sample and makes samples comparable. After choosing the subset a new training loader is created to retrain the model and check if the accuracy has improved.

One challenge of using XAI-based sampling would be that it generally requires a model, which was already trained and has some level of predictive accuracy to generate reliable explanations. For this reason, this sampling technique is more suitable for mid-training adjustments or model retraining than for the initial model training phase.

4.2 Sample Weighting with XAI

Another variant is to use XAI to determine weights for each sample based on each sample's contribution to the model's predictions. This strategy evaluates each sample and weights them so that specific samples have a greater influence on the model's parameter updates.

This weighting can be applied either positively to emphasize samples with high relevance scores or negatively to down-weight highly influential samples and focus on harder lower-relevance samples. This has a two-folded effect. On the one side, positive weighting helps the model learn key features more effectively by prioritizing highly relevant samples. On the other side, negative weighting encourages the model to focus on less confident or more challenging examples which could potentially improve generalization.

4.3 Evaluation of Sampling-based approach

Interestingly, all of the presented methods can be applied to data sampling or sample weighting because they all return a heatmap that indicates the relevance score of each pixel. This score is then evaluated with a chosen metric to rank the samples. However, we can already determine that perturbation-based methods like SHAP or LIME are impractical in this context since even for a single explanation, it requires a large amount of computational resources due to the necessary perturbations. Therefore, generating explanations for all samples in the dataset would counteract our goal of efficient training, although it could potentially improve model accuracy. On the contrary, these sampling methods are feasible with propagation-based methods. With a dataset of size N, all samples can be computed in n backward propagation steps, which is relatively fast compared to SHAP or Lime.

In addition to the computational challenges of perturbation-based methods, gradient-based methods also face critical limitations. They are often prone to issues that lead to inaccurate or noisy attributions, which makes them rather unsuitable for sampling approaches. These limitations include gradient saturation and vanishing gradients.

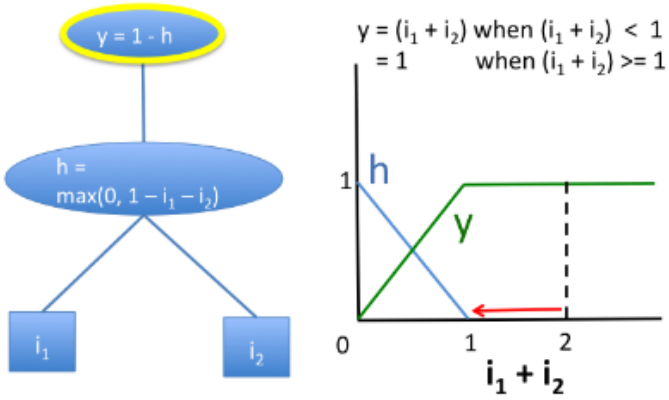

Figure 4.1: Saturation Problem [77]

The gradient saturation problem occurs when the output of a neural network becomes unresponsive to changes in the input. Shrikumar, Greenside, and Kundaje [77] visualized the saturation problem as in Figure 4.1, where a neural network's output y saturates or stops changing, when inputs exceed a certain threshold, in this case, greater than 1. In this saturated state, the gradient becomes zero. This means the network can no longer learn or adapt, as it can't update its weights. This makes it difficult to extract meaningful information about feature importance because the gradients fail to reflect the input's contribution to the output. Also, it implies that changing the input will not have any effect on the outcome, making perturbation-based methods less effective. Gradient saturation is a major challenge

for using XAI in model training since meaningful gradients are essential for guiding the learning process.

The vanishing gradient problem is another critical issue, especially in deep networks. It occurs when the gradients of the loss function become exceedingly small and diminish to near-zero values while they are propagated back through the layers during training [87]. When applying XAI methods that rely on gradients, such as saliency maps or gradient-based attribution techniques, this issue can severely impact their quality. As gradients become extremely small, these XAI methods provide inaccurate or incomplete explanations of feature importance. This hinders the integration of XAI into model training because the model prioritizes irrelevant features or fail to adjust weights effectively.

Independent of the computational barriers of perturbation methods and the limitations of some of gradient-based methods, the biggest crucial challenge lies in identifying a suitable metric for aggregating and comparing relevance scores. The goal would be to find a single value for each sample, making it comparable and sortable. However, using the relevance scores for this is way harder than it seems. Each relevance explanation is essentially just a 2D vector, with each pixel assigned a relevance score. Metrics like the mean, sum, or max of relevance scores may seem logical at first but have inherent limitations. The mean gives us an average relevance per pixel which makes it incapable of recognizing the impact of highly relevant regions if they are few. The sum of relevance scores captures the total relevance across all pixels, which can easily return 0 if the sum of positive and negative attribution is equal. Lastly, using the maximum relevance score focuses only on the single most relevant pixel. This can help identify images with extremely high relevance in small regions but ignores the rest.

When starting working on this thesis, the original objective was to use XAI to create a metric for weighting samples, which shaped the research direction. This would allow for retraining models by weighting the dataset's samples in a retraining pipeline that helps the model focus on significant samples. However, finding a consistent and integrative metric proved out to be challenging. And except for one paper in which researchers developed a useful metric, a unique metric could not be developed, which is why this thesis changed its focus from sample-based methods to integrating XAI directly into the model. The concept of this metric will be discussed in detail in the next chapter.

4.4 Integrating XAI Directly into Model Training

Instead of using XAI to influence the data, another promising approach involves leveraging XAI methods to directly affect the model itself. This more sophisticated strategy integrates an XAI method into the model's inner structure and training process. By using relevance scores generated by the XAI method, the training dynamics can be modified, either by influencing gradient updates or adjusting the model's weights based on the attribution of each neuron to the model's decision. Propagation-based methods assign relevance scores to each neuron which indicate whether it contributes positively or negatively to the classification decision. These scores then guide weight updates or gradient calculations, allowing neurons with higher relevance to have a stronger influence on the decision, while neurons with negative relevance have less impact. This approach essentially "nudges" the model's parameter updates according to the neuron-wise relevance.

A benefit of integrating XAI directly into model training is that we can achieve targeted Gradient Updates. By adjusting gradients based on relevance, this approach can help the model focus on important features, improving the model's generalization. Also, relevance-based optimization has the potential to speed up training by guiding updates along the most relevant paths, leading to higher accuracy in fewer epochs or batches. However, integrating XAI explanations into the training process requires a careful balance to prevent overemphasis on specific features or neurons, which could lead to overfitting. Additionally, this approach may increase the computational complexity of training as well as memory usage. This is because the relevance scores need to be computed, stored and incorporated in real time during each training step.

Among the methods available, only propagation-based methods can be integrated into the model internal structure, as these methods are not model-agnostic and directly utilize the network's structure. Within propagation-based methods, two groups of approaches exist. The first group includes gradient-based propagation methods such as Saliency Maps, DeconvNet, Guided Backprop, SmoothGrad, Gradient * Input, Integrated Gradients, and CAM/Grad-CAM. All of these use gradients to create their explanations. However, because these methods rely on gradients, it is challenging to use them directly for modifying model parameters such as the gradient. Since gradients are already calculated in backpropagation, they cannot be easily used for nudging parameter updates without interfering with the intended flow.

The second group consists of methods that calculate their own dedicated attribution scores. This group includes Layer-wise Relevance Propagation [72] and DeepLIFT [65]. Unlike gradient-based methods, these approaches calculate neuron-wise relevance scores separately from the gradient, which makes it possible to adjust gradient updates or model weights. These relevance-based adjustments take palce in each neuron or between neuron connections to strengthen positive attributions

and weaken negative ones. However, LRP and DeepLIFT differ significantly in their methodologies and suitability for efficient model training.

DeepLIFT compares the output activation of a model for a given input to those of a reference baseline and calculates the difference to determine each neuron's contribution to the model's output. By propagating these contributions back to the input layer, DeepLIFT highlights the most relevant features of the input [65]. Although DeepLIFT provides useful insights, it has a major limitation. It relies on a single baseline, which can heavily influence the results. For example, using a completely black image as the baseline can distort analyses when the input has a mostly black background. In such cases, the model might fail to attribute differences to foreground objects or miss important features. This dependency on a single baseline makes it difficult to establish consistent and reliable outcomes, particularly because DeepLIFT uses relative measures of feature importance instead of absolute measures. And it also makes the prioritization of inputs or features highly variable and difficult.

In contrast, LRP does not require a baseline. Instead, it calculates relevance directly from the network's internal structure by using specific propagation rules to attribute relevance at each layer. This independence from a reference input allows LRP to offer a more straightforward interpretation of the model's decisions without having to define an ideal or neutral baseline.

Another fundamental difference lies in the principle of "relevance conservation" that LRP follows. As relevance scores propagate backward through the network, LRP conserves the total relevance from the output layer, distributing it across previous layers in such a way that the total relevance remains consistent. This conservation principle ensures that relevance scores add up accurately across layers and preserve a balanced view of attribution without a baseline. This characteristic makes LRP particularly valuable to integrate it into the internal training process because it provides well proportionally scaled relevance values that effectively can nudge the model parameters

Given these differences, LRP may be more suitable for integration into model training where relevance scores are used to adjust weights or gradients directly. Its independence from baselines and strict conservation of relevance allow it to be applied consistently across samples, maintaining stable interpretations that align closely with the model's structure. In contrast, DeepLIFT's dependence on a baseline can lead to inconsistencies and complicate integration into training processes, as the relevance scores may vary with different baseline choices.

The following table summarizes the most important attributes from the previous discussion. It becomes clear that LRP is the most suitable method for direct integration, especially because of its absolute attribution values and the conservation of relevance.

Method	Model Agnostic	Model Specific	Perturbation-based	Surrogate-based	Propagation-based	Gradient-based	High Comp. Effort	No baseline	Applying into model structure	Independent attribution relevance	Conservation of Relevance
LIME	X		X	X			X				
SHAP	X		X				X				
DeepSHAP		X	X		X	X	X				
Saliency Maps		X			X	X		X			
DeconvNet		X			X	X		X			
Guided Backprop		X			X	X		X			
Gradient * Input		X			X	X		X			
Integrated Gradients		X			X	X	X				
CAM / Grad-CAM		X			X	X		X			
DeepLIFT		X			X				X	X	
LRP		X			X			X	X	X	X

Table 4.1: Comparison of XAI Methods

5 Previous Work

Before we introduce how LRP is integrated into model training, it is essential to review previous research that has utilized explainable AI for efficient model training.

In chapter 4, we discussed whether a metric-based sample approach could be a useful approach to attain efficient model training. However, we came to the conclusion that it is problematic to find a metric that evaluates the relevancy values effectively. This challenge has been addressed in a key paper, which will be presented below.

5.1 SHAP-Based Sample Weighting

In October 2022 researchers from Infineon Technologies AG and the universities of Erlangen and Munich proposed a retraining pipeline utilizing Explainable AI for improving the performance of Neural Networks [4]. Their work was designed for radar-based people-counting scenarios and leverages SHAP values to create a metric that assigns weights to training samples based on their difficulty. This metric forms the foundation of an incremental retraining process that aims to improve the robustness and precision of the model. The core contribution of the paper is the development of two methods which determine sample weights based on SHAP values.

The first method called **Masked Difference** concentrates on identifying specific features that positively contribute to incorrect predictions. The approach begins by masking these features which takes out their influence on the prediction. It then computes the difference in SHAP values between the model's incorrect and correct predictions. This difference is used to determine how much the weight of a sample should be adjusted. By emphasizing samples where incorrect predictions result from specific feature contributions, the method ensures that the model pays more attention to these challenging cases during retraining. This technique effectively prioritizes samples that are inherently harder for the model to classify correctly. This makes them more impactful in shaping the model's learning process.

The second method, **Local Difference**, takes a slightly different approach by focusing on the pixel-wise differences in SHAP values between incorrect and correct predictions for each feature. For every feature in a sample, the method computes how

much the SHAP value differs between these two types of predictions. These individual differences are then aggregated into a single measure, which serves as a metric and reflects the overall difficulty of the sample. Larger aggregated distances signify more complex or challenging samples, which are subsequently assigned higher weights. Like the previous metric, this method ensures that samples with greater predictive difficulty play a larger role in the training process. Unlike the Masked Difference approach, which emphasizes specific features contributing to errors, Local Difference provides a more overall measure of sample difficulty based on the overall contribution of all features. The weight of each sample is updated before retraining, ensuring that harder-to-predict samples influence the training process more significantly.

The retraining process starts by dividing the data into a training set (D_m), validation set (D_v), and test set (D_t). During training, the model is evaluated on subsets of D_t to identify incorrectly predicted samples. These misclassified samples are grouped into incremental datasets ($D_{I1}, \ldots, D_{Is}$) for retraining. SHAP values are computed for the misclassified samples, and this metric is used to adjust their weights. The model is then retrained iteratively on the combined dataset, with weights updated at each step to refine the model's predictions.

The final result of this methodology was that their pipeline method could lead to an increase of the accuracy performance of their people-counting model by 4%. Performing their method on the CIFAR-10 Dataset with the pre-trained ResNet-18 model led to an increase in performance by 3%.

5.2 LRP for adaptive learning rate

In chapter 4 we concluded that LRP is very much in question to be applied directly to model integration. This idea was applied in the paper we are going to discuss next.

In October 2019, researchers from the University of Science and Technology in Korea published a paper on utilizing Layer-Wise Relevance Propagation to optimize the learning process during the training of deep neural networks [5]. Unlike traditional optimization techniques, their method uses LRP to measure the influence of each neuron on the network's output and incorporates this information into the weight adjustment during training.

Traditionally, the weights in a neural network are updated using the following equation, which represents one step of stochastic gradient descent:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W_{\text{old}}}$$

Here, η is the learning rate, and $\frac{\partial L}{\partial W_{\text{old}}}$ is the gradient of the loss function L with respect to the weights W. This equation encapsulates the fundamental mechanism

which deep learning models use to iteratively reduce the prediction error by adjusting the weights between neurons. Optimizers like Adam or AdamGrad build upon this basic principle. What makes them special is that they incorporate mathematical heuristics to approximate the optimal gradient steps to provide efficient and robust training.

The innovation in the Korean researchers' approach lies in the integration of LRP scores, denoted as β, into this weight update equation. By multiplying the gradient term $\frac{\partial L}{\partial W_{\text{old}}}$ with the relevance score β, the updated equation prioritizes neurons based on their contribution to the network's output:

$$W_{\text{new}} = W_{\text{old}} - \eta \frac{\partial L}{\partial W_{\text{old}}} \beta \tag{5.1}$$

This modification shifts the optimization process from a purely mathematical heuristic to one that incorporates explainability insights. It is also important to emphasize that this novel approach does not use the post-hoc idea of XAI to explain the model's decisions after training. Instead, it integrates the neuron-wise calculated influence on the decision directly into the update of each neuron's weight, thereby embedding XAI directly into the training process.

In their testing results, the researches demonstrated that the LRP-based optimization methods showed better or comparable performance to traditional methods like Stochastic Gradient Descent (SGD), Adam, AdaGrad, and AdaDelta. However, there were some challenges with memory overhead due to the additional calculations required by the LRP-based methods. The final result of the paper was that integrating LRP into the training process can significantly improve the performance and confidence of neural network predictions. However they could not surpass already established optimizers what can be seen in the table.

	MNIST		CIFAR-10	
Method	SPI	accuracy	SPI	accuracy
SGD	8.72	94.7	13.75	76.57
Adam	8.93	97.8	14.71	77.72
AdaGrad	8.75	97.8	14.04	77.98
Adadelta	9.35	93.2	14.24	73.94
lrp_invert	9.19	94.8	118.39	76.26
lrp_Eucdist	9.04	96.2	121.37	74.3
lrp_sgd_v4	9.18	95.8	121.46	76.21

Figure 5.1: LRP optimizer results

5.3 Transition to Thesis Work

While the previously presented paper presents an innovative use of LRP, its proposal to use LRP as a standalone optimizer introduces unnecessary complexity, given that already almost optimal and well-established optimizer algorithms like Adam exist. Adam is highly efficient and widely regarded as one of the most effective optimizers for training deep networks. Instead of replacing Adam or other optimizers, this thesis work focuses on combining the strengths of LRP with these existing methods. Specifically, we propose using LRP alongside a conventional optimizer by integrating neuron relevance into the gradient computation. Rather than completely reengineering the optimization process, this approach involves nudging the gradient updates with LRP relevance values, offering a simpler method to enhance training performance.

By refining the training process through this combined approach, our goal is to leverage the strengths of LRP to further optimize weight updates while retaining the proven benefits of robust optimizers like Adam. The fundamental difference of this approach to the paper is that the paper computes the gradients and redirects them with the attribution values afterwards, while the proposed approach already applies the attribution values one dimension deeper, directly inside the computation of the gradients. This method not only avoids the drawbacks of creating a new standalone optimizer but also provides a practical way to enhance the efficiency of existing training processes.

6 Methodology

This chapter addresses how Layer-wise Relevance Propagation (LRP) can be integrated into model training to improve performance. First, the concept of how to integrate LRP will be explained, followed by a discussion on the implementation of the LRP algorithm in PyTorch. Then the process of implementing LRP integration into model training is presented. Finally, the used models and the datasets which were used to test this approach will be presented, as well as how the experimental and hardware setup looked like.

6.1 Integrating LRP into odel training and applying it on the gradient

The main idea introduced in this thesis is the application of LRP directly into the internal network training structure. As discussed previously, the process of LRP is to propagate the attribution from layer to layer, starting on the classified output neuron and from there reaching back until the input image. In fact, the relevance value is computed for each neuron in each layer. This granular procedure enables us to view the attribution of each single layer to the neuron with the highest probability in the output layer. The proposed method is now to apply each neuron's relevance value on its own gradient. To achieve this, both the gradient and the relevance propagation are computed simultaneously, by which the neuron-specific relevance can be directly applied to its corresponding gradient. This separates the approach from the standard training as well as the traditional LRP method because the standard procedure usually is to train the model first and then use LRP to generate an explanation. Now, LRP is used during training inside the gradient computation not to generate an explanation, but to improve the performance.

6.1.1 Expected Effect of LRP-Enhanced Training

The expected effect of integrating LRP into the training process is to improve the efficiency and accuracy of model training. Therefore, this approach aims to demonstrate that the same level of accuracy can be achieved with less training data by guiding the learning process based on neuron relevance and allowing the model to focus on the most informative features.

6.1.2 Effect on Gradient-Based Weight Updates

In standard backpropagation, the weight update for a neuron i in layer l during each epoch is calculated as:

$$\Delta w_i = -\eta \cdot \frac{\partial L}{\partial w_i}$$

where η is the learning rate, L is the loss function and $\frac{\partial L}{\partial w_i^l}$ is the gradient of the loss with respect to the weight w_i.

In LRP-enhanced training, we modify the gradient by introducing a relevance-based adjustment. The relevance value for each neuron R_i^l, which represents the contribution of that neuron to the output decision, is applied to its corresponding gradient. This leads to the following updated weight modification rule:

$$\Delta w_i^l = -\eta \cdot \left(\frac{\partial L}{\partial w_i^l} \cdot (1 + \lambda R_i^l) \right)$$

where R_i^l is the relevance score for neuron i in layer l and λ is a scaling factor that controls the influence of the relevance score on the gradient.

By applying this adjustment, we aim to enhance the contribution of neurons with higher relevance scores and reduce the influence of less relevant neurons. This modification should focus the model's learning on the most important parts of the data, thus improving the final accuracy of the model.

The reason the relevance score is not directly multiplied with the gradient, but instead with a λR_i^l factor, is to carefully control the effect that the relevance has on the gradient. Also, the addition of 1 ensures that the relevance values, which typically range around 0, can both increase or decrease the gradient depending on whether the relevance is positive or negative. Otherwise, an attribution value of 0.01, which would actually indicate a positive attribution, would increase the gradient massively if multiplied directly. Using the +1 ensures that the gradient gets increased. This is similar to adjusting the gradient by a percentage. With an attribution of 0.01, meaning 1%, the gradient gets increased by 101%.

The scaling factor λ is introduced to regulate the overall influence of the relevance on the gradient. Without this factor, the relevance could potentially cause large fluctuations in the gradient, leading to instability in training. By adjusting λ, it can help ensure that the relevance scores contribute to the gradient in a controlled manner and prevent the gradients from exploding to excessively high values, which could disrupt or destabilize the training process.

6.1.3 Visualizing Expected Effect

The goal is to show that using less training data with LRP training can perform to the same accuracy compared to standard model training using more data. To visualize this effect, let a coordinate system contain the accuracy on the y-Achsis and the percentage of training data being used on the y Achsis.

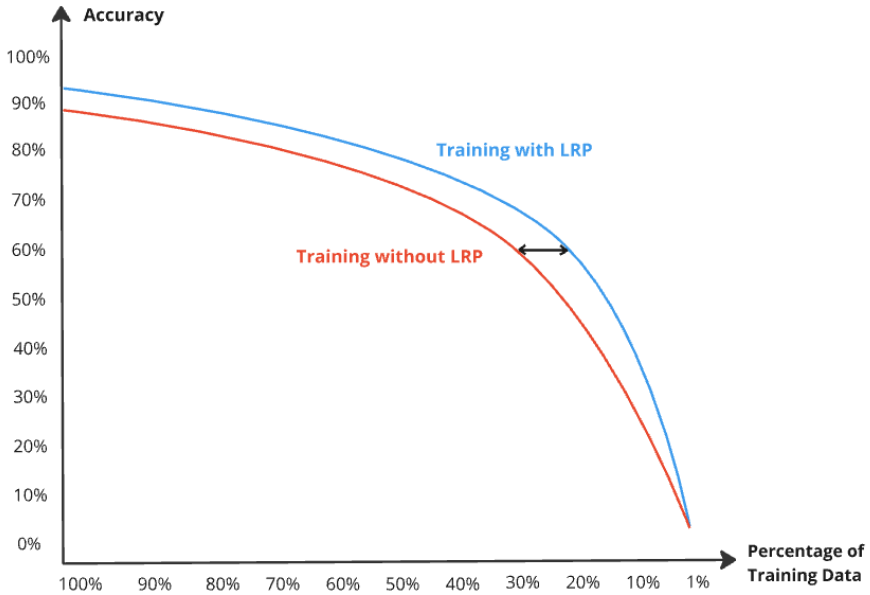

Figure 6.1: Ideal Effect

Ideally, the LRP accuracy line which is shown in blue lies above the standard training accuracy line. By taking a horizontal black line between the two curves, it can be measured how much less training data is sufficient to provide the exact same accuracy compared to not using LRP integrated training.

6.1.4 LRP Implementation and Usage

In theory, computing the relevance score in each layer and passing it to the next layer is not overly complex. The LRP rules are basic arithmetic operations and can be coded in a few lines of code. In addition, computing LRP values is possible just in the time of one backward pass and includes fewer operations than a standard gradient-based backward pass. To illustrate, LRP simply divides the output relevance proportionally based on each input's contribution, such as distributing the relevance by the sum of weighted inputs. In contrast, a normal backward pass requires computing partial derivatives for every parameter, applying the chain rule, and updating the weights according to those derivatives, which results in additional computational steps. This makes LRP computationally less intensive.

However, pythons most renowned libraries for machine learning and computer vision, Torch and TensorFlow do not natively support layers that compute LRP relevance values. This leads to the necessity of implementing special designed layers that integrate LRP into their backward pass next to the standard backward process. More specifically, wrapper classes can retain the standard forward- and backward-propagation functionality, but also provide the possibility of adding overhead functionality. This overhead functionality is useful, as we see later, to further integrate LRP directly into the model's internal process.

The conceptual approach for implementing LRP was inspired by Frederik Hvilshøj's TorchLRP repository Github. However, the specific implementation presented in this thesis was developed independently to address the unique requirements of integrating LRP directly into the training process.

For each layer we introduce three wrapper classes, $LinearLRPLayer$, $ConvolutionalLRPLayer$ and $MaxPoolingLayer$ that wrap around PyTorch's standard layers and split between the usual torch class or the LRP class. This split depends on if the standard linear backward pass or if the LRP backward pass should be applied and is realized via a conditional clause as visualized in image 6.2.

Each of these layers has two modes of operation. First, they can perform the standard forward and backward pass, which is used during regular model training. This process is identical to PyTorch's standard propagation and requires no additional modifications. Second, they can apply LRP in the backward pass next to the standard backward pass. To achieve this, specialized classes that override PyTorch's default forward and backward methods were created.

Let them be $LinearLRP$, $ConvolutionalLRP$ and $MaxPoolingLRP$. Each of them contains it's own forward and backward method. The forward method behaves similarly to PyTorch's standard implementation, with the primary role of propagating the input through the layer. However, in addition to this, the forward pass also saves key parameters, specifically, the input, weight, and bias. These are critical for the later computation of relevance in the backward pass. This additional step can

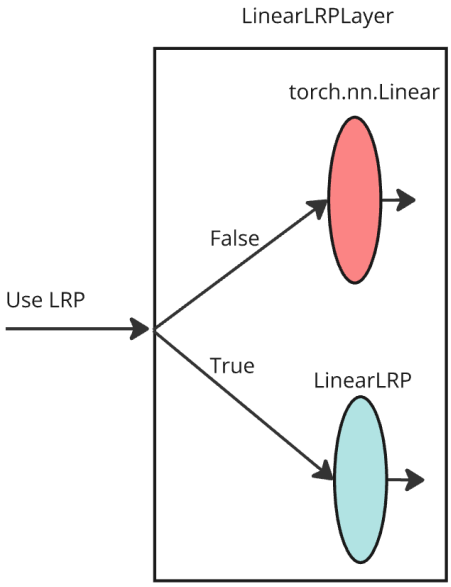

Figure 6.2: Wrapper

be enforced by using pytorch's context property, which enables storing additional layer-specific data during the forward pass. This ensures that these values can be accessed during the backward pass.

The primary application of LRP takes place inside the backward layer. Instead of computing the gradient and passing it to the previous layer, LRP is applied to compute the relevance distribution and propagate this relevance backward through the network. Let *apply_LRP_rule* be the function to compute the attribution values within a given layer l_i, then this function uses the input, weight, bias and the gradient output of the current layer to calculate the attribution. These parameters, input, weight, and bias, were previously saved during the forward pass. The input refers to the activations of neurons from the preceding layer l_{i-1} which are passed into the current layer l_n. The weights represent the parameters connecting the neurons between layer l_i and l_{i-1}, defining the strength of the influence exerted by each input neuron on the subsequent layer's output. The bias is an additional parameter and is incorporated into the neuron's activation in each layer to modulate the output alongside the weighted sum of the inputs.

In the backward pass of any LRP computation, the initial input is the gradient of the output of the current layer, also referred to as *grad_output*. This gradient reflects the influence of the current layer on the final output, and serves as the basis for relevance propagation. We can then compute the attribution values based on the chosen LRP Rule.

The typical process of using LRP to generate explanation images or relevance heatmaps begins with training the model using standard PyTorch layers (e.g., nn.Linear) until a stopping criterion is met. To explain a given input after the

model was trained in the end, we use the wrapper's ability to switch the model's layer functionality to LRP-based layers and perform a forward pass through these layers. This forward step is similar to pytorch's nn.linear forward step. The crucial difference is now the backward process, in which the selected LRP rule is applied on the layers one by one until the input layer. This is visualized in image 6.3. The resulting tensor, which has the same shape of the original input image, can be used to generate a heatmap showing the attribution to the decision. The ideal result for images would be that these pixels which represent the number, have a higher attribution than the black background pixels.

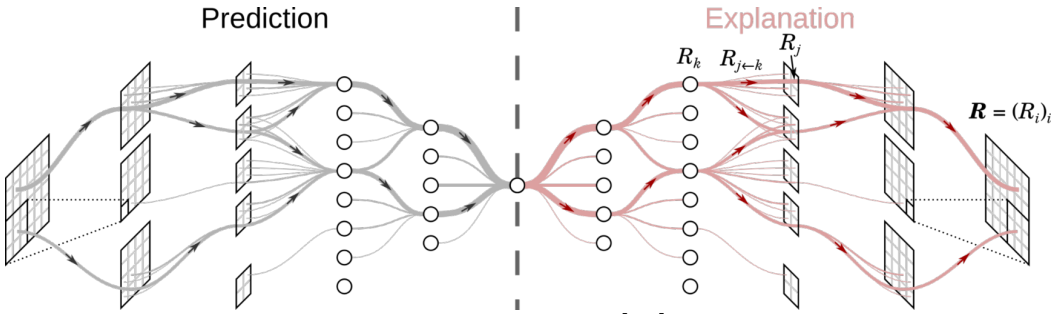

Figure 6.3: LRP [83]

This chapter was about how LRP can be implemented to generate an explanation for a given image. The next chapter discusses how LRP can be integrated into model training.

6.1.5 Implementation of LRP Integration into Model Training

Using Wrapper around the model layers comes in handy for integrating LRP, as it allows to add further functionality to the forward as well as to the backward methods. This enables to adjust the gradients by incorporating relevance scores, which is the primary objective of the approach.

During the backward pass, the process begins at the output layer. The initial backward step receives the neuron values of the output layer, which correspond to the predicted probabilities for each class. The selected LRP rule is applied to redistribute this relevance to the preceding layer, calculating a relevance map aligned with the shape of the preceding layer, where each neuron's relevance score reflects its contribution to the final prediction. This relevance value tensor has the shape of the preceding layer, which is (number of neurons in layer, batch size). This relevance map is cached using `set_relevance_cache` for use in the next backward step.

In the output layer, the relevance values are not immediately applied to the gradient. Instead, the usual gradient computation for the output layer takes place, leaving the output layer's gradient unaffected by relevance. The relevance map computed here is simply cached and will be applied in the subsequent layer.

Moving to the next layer which is now the preceding layer relative to the output layer, the cached relevance from the previous step gets retrieved. This relevance score is applied to the gradient of the current layer by scaling it with $(1+\lambda \times R_{\text{prev}})$, where λ is a scaling parameter, and R_{prev} contains the relevance values from the preceding layer. This adjustment modifies `grad_output`, ensuring each layer's gradients are influenced by relevance information from the layer above. After modifying the gradient, the relevance score are recomputed for the current layer using `apply_lrp_rule` and cached for the following backward step to apply them there in the next step.

Since model training takes place in batches, this relevance-based gradient adjustment operates across both neurons and the batch dimension, represented as (neuron in layer, batch size). This enables to compute relevance for each prediction in the batch, indicating each neuron's specific contribution to each decision within the batch.

This iterative process continues through each layer when moving backward through the network. At each step, relevance scores are recalculated, cached, and used to adjust gradients in the subsequent layer. An error check on the relevance cache ensures the propagation is consistent and accurate across layers.

For intermediate layers, the adjusted gradients (`grad_input`, `grad_weight`, `grad_bias`) are computed based on modified `grad_output`, enabling relevance-weighted updates to the model parameters during training. This allows each layer's gradient updates to reflect neuron influence, achieving a more refined gradient update aligned with the neurons' contributions.

By the time the input layer is reached, each layer's gradients have been modified according to their respective relevance scores, yielding an adjustment across (neuron in layer, batch size) for each prediction in the batch.

The following code snippet describes the training process of the presented approach and summarizes what has been introduced earlier. Model training in general always starts by iterating through all batches that make up the training data in each epoch. For each batch, the process starts with a forward pass, where the model output O is computed. With the output O and the true labels, the loss L can be computed.

Next, the backward step begins by checking whether the current layer is the output layer. In the output layer, the gradient for the weights connected to the preceding layer is calculated, along with the attribution values between the output layer and the preceding layer. These attribution values are stored to be applied to the gradient of the subsequent layers.

If the current layer is not the output layer, the cached relevance values between the current layer and the preceding layer are retrieved, and the gradients are updated using the LRP-enhanced attribution values. The weights are then adjusted accordingly.

Algorithm 1 Model Training with Layer-wise Relevance Propagation (LRP)

for each epoch **do**
  for each batch (X_b, Y_b) in X **do**
    Forward Pass:
    $O \leftarrow M(X_b)$ ▷ Compute model output
    $L \leftarrow \text{Loss}(O, Y_b)$ ▷ Compute loss
    Backward Pass:
    if Layer is Output Layer **then**
      $R_L \leftarrow \text{LRP}(\text{Output Layer}, \text{rule} = r)$ ▷ Apply LRP to output layer
      $Cache \leftarrow R_L$
    else
      $R_{\text{next}} \leftarrow$ Retrieve Cache
      $\text{grad} \leftarrow \text{grad} \times (1 + R_{\text{next}})$ ▷ Modify gradients with relevance
      $R_i \leftarrow \text{LRP}(\text{Current Layer}, \text{rule} = r)$ ▷ Cache new relevance
    end if
    Update Weights:
    $M \leftarrow M - \eta \times \text{grad}$
  end for
end for

6.1.6 The Model and the Datasets

To analyze the effects of LRP, two models were created. The first model is a fully connected linear neural network. This model will be tested on the MNIST dataset. MNIST is one of the most known datasets in machine learning and has been widely used as a benchmark for evaluating models. The dataset consists of 70,000 grayscale images of handwritten digits, ranging from 0 to 9 and it is divided into a training set of 60,000 images and a test set of 10,000 images. MNIST owes its popularity to its remarkable properties: The images are in a uniform format, centered, and size-normalized and own a balanced class distribution. Therefore, MNIST suits perfectly for the development of proof-of-concept models and demonstrating the effectiveness of new methods.

The second model is a convolutional network composed of several convolutional, maxpooling, and linear layers. This model is tested on CIFAR-10. The CIFAR-10 data set consists of 60,000 images evenly distributed across 10 distinct classes, such as airplanes, automobiles, birds, cats, and dogs. Each class has 6,000 images, with 5,000 images per class in the training set and 1,000 images per class in the test set. CIFAR-10 shares similar properties with MNIST, with the key difference being that its images are represented in a RGB format with 3 color channels (Red, Green, and Blue). This makes CIFAR-10 the ideal benchmark data set for convolutional model experiments.

The network structures of both multilayer and convolutional networks that were used in this thesis are shown in Table 6.1

Multi-layer neural network	**Deep Convolution neural network**
Input(32x32x1 image)	Input(32x32x3 image)
Fully Connected-1024	Conv 3-64
Fully Connected-512	Conv 64-64
Fully Connected-10	Maxpool
	Conv 64-128
	Conv 128-128
	Maxpool
	Conv 128-256
	Maxpool
	Fully Connected-1024
	Fully Connected-512
	Fully Connected-10

Table 6.1: The two architectures used in the experiments.

6.1.7 Experimental Setup

In this section, the experimental setup that was used to evaluate the effects of Integrating LRP on model training will be described.

The experimental design aims to maintain fairness and consistency across different models and configurations. This allows a clear comparison between LRP-enhanced training and standard training. The experiments are divided into two main parts. First, the LRP integration is evaluated on the fully connected linear network using the MNIST dataset. Next, the same setup is applied to the convolutional network with the CIFAR-10 dataset.

In each experiment, two types of models gets compared, one that integrates LRP and one that uses standard training without LRP. To validate the efficiency gains of using LRP, both models are tested on progressively smaller subsets of the dataset which range from 100% down to 1 %. For each data percentage, both models are trained and the final accuracy results are compared. The trick for generating valid comparison data is to provide the same test conditions for each model and each percentage. This includes taking out any randomness which can occur during training.

By controlling key parameters, such as the number of epochs, dataset size intervals, the choosen data subset, LRP integration factor and random seeds, it is ensured that any observed differences in performance can be attributed to the effects of LRP, rather than other variables or effects. Setting the seed is essential for fair comparisons

between different experimental setups. It ensures that all models begin with the same initial conditions. When a random seed is set, processes like initializing model weights, shuffling data, or any other action that involves randomness will follow the same sequence of "random" choices each time the code is run. This also means that it selects the same subset of data for each percentage. This consistency allows to isolate and reliably attribute performance differences to the experimental variable of interest, the integration of LRP, rather than to random variations.

Finally, to enhance the reliability of our results, each training process is repeated multiple times for each percentage and the accuracies are averaged to further reduce the impact of any random variations. Herefore it is important to emphasize that the same initial seed for each percentage are used, but the seed get incremented by one for each repeated training run. This approach ensures that the random seeds within each set of runs are altering but also remain consistent across all percentages. This ensures a fair comparison and preserves consistency throughout all experiments. The flowchart 6.4 visualizes this process.

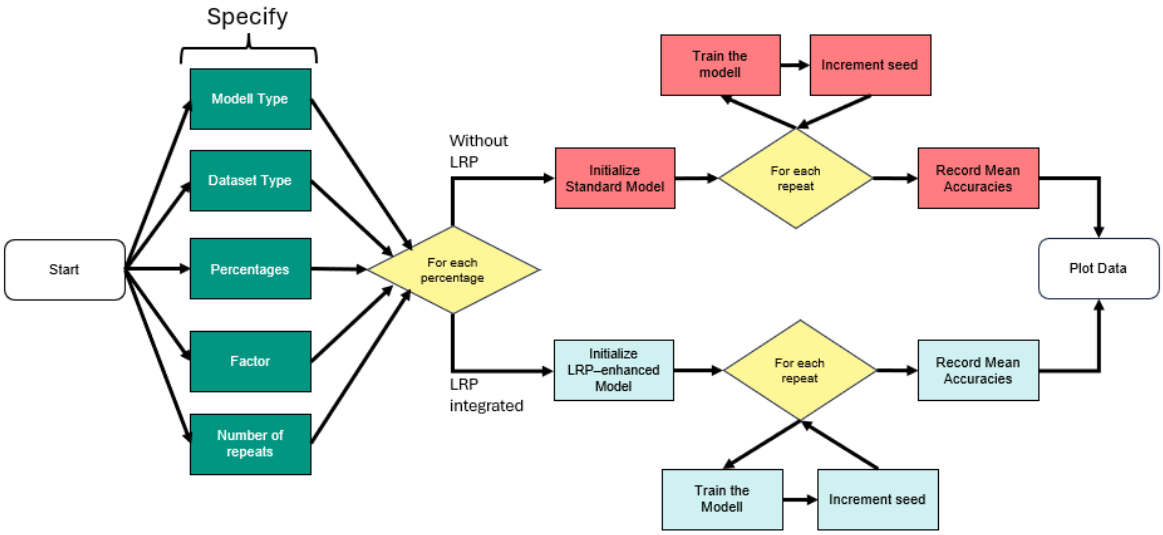

Figure 6.4: Experiment Flowchart

6.1.8 Physical Setup

This section describes the physical setup used to conduct the experiments from the previous section to relate the computational time measurement to the specific hardware specifications.

The experiments were executed on a machine equipped with an NVIDIA GeForce RTX 3050 GPU, featuring 4 GB of VRAM. The machine was also equipped with 16 GB of RAM and an 11th Gen Intel(R) Core(TM) i7-11800H processor with 8 physical cores and 16 threads.

The training and evaluation processes were implemented in Python 3.12.7, utilizing PyTorch 2.3.1 for model definition and training. The CUDA toolkit version 12.7 was employed to leverage GPU acceleration during computations, while the NVIDIA driver version 565.57.01 ensured optimal hardware compatibility.

All experiments were conducted in an isolated Python virtual environment with identical library versions across runs. The necessary libraries and their versions can be taken from the requiremnts.txt provided in the repository. This setup ensured that the observed differences in model performance were solely attributable to the integration of LRP and not to hardware or software inconsistencies.

7 Results

This chapter examines the results to evaluate the effectiveness of integrating LRP into model training. Before presenting the main experiments, it is worth briefly discussing the first valid results achieved during the implementation and initial testing of our approach. These preliminary results provided a clear indication that incorporating LRP into model training can positively impact model accuracy and this initial success served as a strong motivation to further investigate and refine this method.

Figure 7.1 illustrates the outcomes of a small-scale experiment designed to test both the implementation and the potential impact of LRP integration. The experiment involved 15 consecutive training runs, comparing two models, one with LRP integration and one without. Both models were trained using 5% of the dataset for 3 epochs in each run. Although the seeds for each run were random, they were identical across both models to ensure a fair comparison. The results showed that the model trained with LRP integration achieved an average final accuracy of 91.00%, while the model without LRP reached an average final accuracy of 90.49%. Although the improvement in accuracy was relatively small, it demonstrated a consistent benefit. However, it is important to note the computational overhead introduced by LRP. The average training time per run with LRP was approximately 8.0 seconds while it was only approximately 4.7 seconds without. Therefore the training with LRP takes significantly longer than training without LRP, nearly 70% more time per run.

Figure 7.1: First Experiment

7.1 Linear Network

7.1.1 Linear Network with LRP factor 1

The first experiment was conducted on the linear network with an LRP factor of 1. The left plot illustrates the results across the full percentage range of training data (100% to 1%), while the right plot provides a more detailed view of the narrower range (30% to 1%). From the right plot a slight improvement can be seen in the accuracy when LRP is integrated. However, the maximum observed improvement is modest and amounts to only 0.26 percentage points. This suggests that LRP integration may have a limited but consistent impact, particularly when training data is reduced. The integration appears to slightly guide gradient updates in a more meaningful direction, improving learning efficiency under low-data conditions. However, with an LRP factor of 1, this effect is minimal and becomes noticeable only when training data is limited.

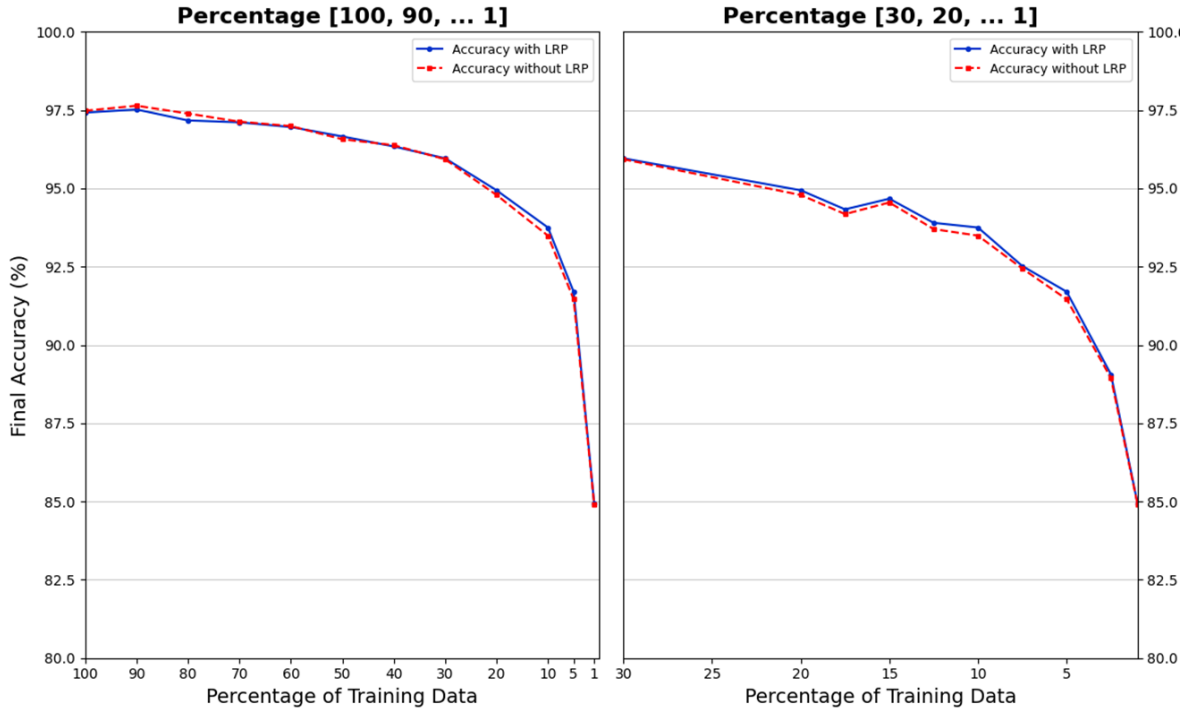

Figure 7.2: NN Factor 1

In addition to accuracy improvements, the training time was also analyzed to assess the computational cost of integrating LRP. As shown in table 7.1, training with LRP took approximately almost three times longer than training without LRP across all percentages of training data. This additional computation time is attributed to the extra steps required to calculate and propagate relevance scores during the backpropagation process. To further explore the results, the next parts will investigate how the impact varies with different LRP factors.

Table 7.1: Training Times with and without LRP (in seconds)

Training Data (%)	Time with LRP (s)	Time without LRP (s)
100	96.43	30.35
90	90.39	27.99
80	80.89	25.39
70	71.45	22.81
60	61.60	20.03
50	53.78	18.21
40	43.62	15.52
30	33.94	12.94
20	24.42	10.24
10	14.85	7.03
1	10.09	5.60

7.1.2 Linear Network with LRP factor 2

Using the same experimental setup, but with the LRP factor doubled to 2, a noticeable improvement in accuracy across almost all data percentage points can be observed. The results demonstrate that increasing the LRP factor can amplify its positive effect on accuracy. However, the maximum observed improvement remains modest with only 0.39 percentage points, while the average improvement across all percentages is just 0.15 percentage points.

These findings indicate that although a higher LRP factor slightly enhances the impact of LRP integration, the overall effect on accuracy remains minimal. This suggests that the LRP factor influences the gradient updates more effectively but does not lead to significant gains in performance. Further exploration is necessary to determine whether larger factors or alternative configurations might yield more substantial improvements.

Stats:
Linear Network
Number of Epochs: 3
Number of Repeats: 4
Factor: 0.2

(%)	Avg. Acc. with LRP (%)	Avg. Acc. Standard (%)	Diff
100%	97.14	96.95	✓
90%	97.45	97.31	✓
80%	97.26	97.37	✗
70%	97.14	97.11	✓
60%	97.07	96.79	✓
50%	96.23	96.25	✗
40%	96.33	96.06	✓
30%	96.01	95.62	✓
20%	94.92	94.76	✓
10%	92.99	92.83	✓
5%	91.14	90.90	✓
1%	81.39	81.35	✓

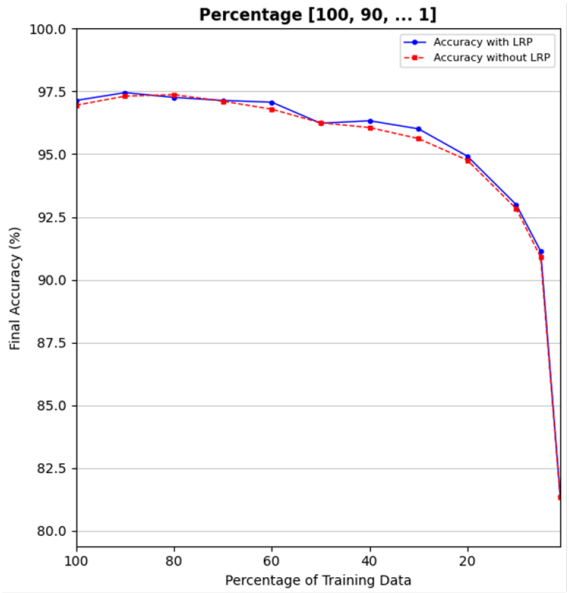

Figure 7.3: Linear Network with factor 2

7.1.3 Linear Network with LRP factor 10

The next experiment investigates the effect of increasing the LRP factor to 10. As shown in Figure 7.4, the accuracy of the LRP-enhanced method initially surpassed that of the standard model. The right plot provides a closer view of the accuracy across percentages ranging from 30% to 1%. The maximum observed improvement was 0.78%, with an average improvement of 0.43% across all percentages. This demonstrates a enhancement compared to factors 1 and 2, but again only for smaller training data percentages. However, when more than 30% of the training data was used, the accuracy dropped drastically as shown on the left plot. The possible reason for this accuracy drop will be discussed next.

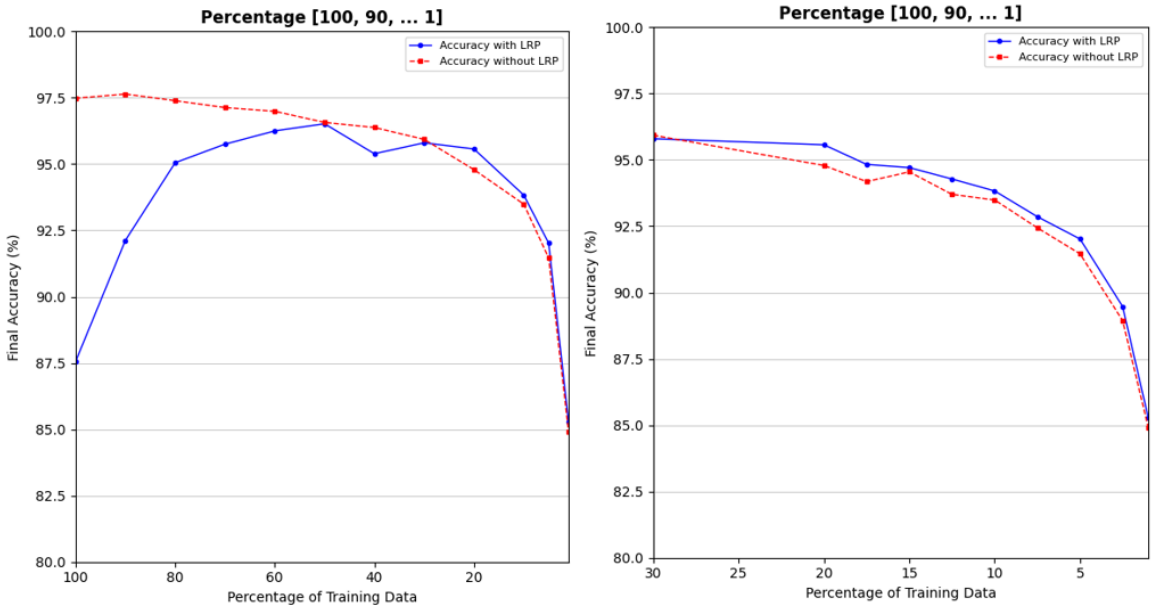

Figure 7.4: Linear Network Factor 10

When looking deeper into the batch-wise progression during training, it reveals that the accuracy decreased sharply after a certain point during each training process. This suggests that scaling the gradients must have a negative effect on the accuracy after a certain training amount is reached. But why does this occur?

The effect occurs earlier when the amount of training data is bigger. When thinking about the increase of training data, it implies that the training process is repeated on more batches. That is because in each epoch, the number of batches is determined by dividing the training dataset size by the batch size. So if the training data percentage increases, the number of batches grows linearly. This means that the more data is used, the more batches are trained and the more often the model's gradients are scaled with the attribution values. As a reminder, the model scales neuron-wise gradients with the formula: $gradient = gradient \cdot (1 + factor \cdot attribution)$. While

this mechanism aims to guide the network's learning process by enhancing relevant neurons and suppressing irrelevant ones, it introduces an accumulation effect. And when increasing the data, this accumulation is stronger as the scaling of the gradient is repeated more often then with lower percentages and amplifies the accumulation over time. This brings two significant risks.

First, as more batches are processed, the repeated gradient adjustments can accumulate over time and lead to gradient explosion. This accumulation causes the gradient values to grow uncontrollably and destabilizes the training process. This instability is more likely when larger datasets are used because the amount of accumulation increases with the amount of data. Also, this results in a collapse in accuracy after processing more than 30% of the data.

Second, for neurons with consistently negative attributions, the adjustment mechanism progressively reduces their gradient values which potentially causes gradient vanishing. In this scenario, gradients become too small to contribute meaningfully to weight updates. This can effectively freeze parts of the network and limit the network's ability to learn effectively, especially in deeper layers of the network.

The choice of factor used to scale the gradients plays a critical role in the stability of the training process. If the factor is too large, it increases the risk of overfitting, which becomes even more likely when the amount of training data is high.

7.2 Convolutional Network

After analyzing the results for the Neural Network and MNIST, the results for the convolutional network and Cifar-10 will be analyzed next.

7.2.1 Convolutional Network Factor 0.1

Unfortunately, the first results for the CNN with an LRP factor of 0.1 were underwhelming. The results shown in Figure 7.3 indicate no improvements at all. In fact, the performance deteriorates, as it worsens significantly after 4 epochs, as illustrated in Figure 7.6. Taking a closer look at the results for higher percentages, the effect also appears to be counterproductive as the performance dropps further when more training data is used. A possible explanation could be the same issue discussed in section 7.1.3, where gradient explosion and gradient vanishing likely occurs, leading to overfitting and destabilized training. Lowering the factor further closer to zero, like in Figure 7.5, results in more stable gradients. In such cases, overfitting only happens in later epochs. However, despite these adjustments, no measurable improvement in performance could be observed. This suggests that using an LRP factor of 0.1 and lower for CNNs not only fails to enhance accuracy but may also introduce instabilities that negatively impact the training process.

Stats:
CNN Model
Number of Epochs: 3
Number of Repeats: 4
Factor: 0.1

(%)	Avg. Acc. with LRP (%)	Avg. Acc. Standard (%)	Diff
100%	72.08	73.27	✗
90%	65.95	72.8	✗
80%	71.47	70.68	✓
70%	69.08	69.51	✗
60%	66.67	67.17	✗
50%	65.58	65.38	✓
40%	61.57	62.18	✗
30%	57.12	56.72	✓
20%	50.65	50.45	✓
10%	40.15	39.47	✓
5%	32.02	32.67	✗
1%	19.76	18.63	✓

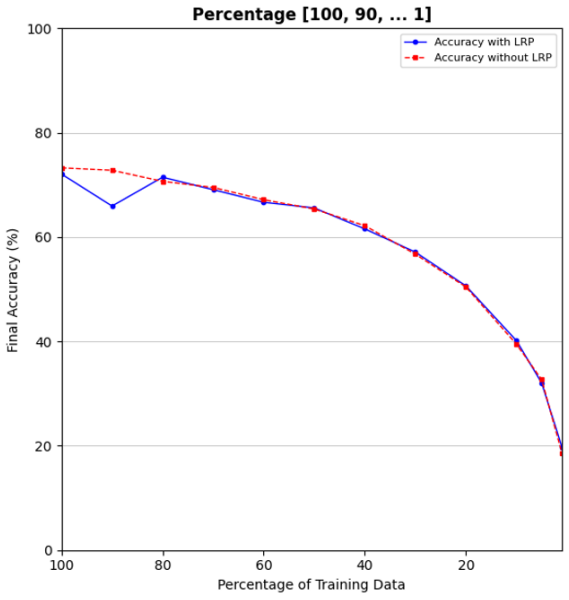

Figure 7.5: CNN with factor 0.1

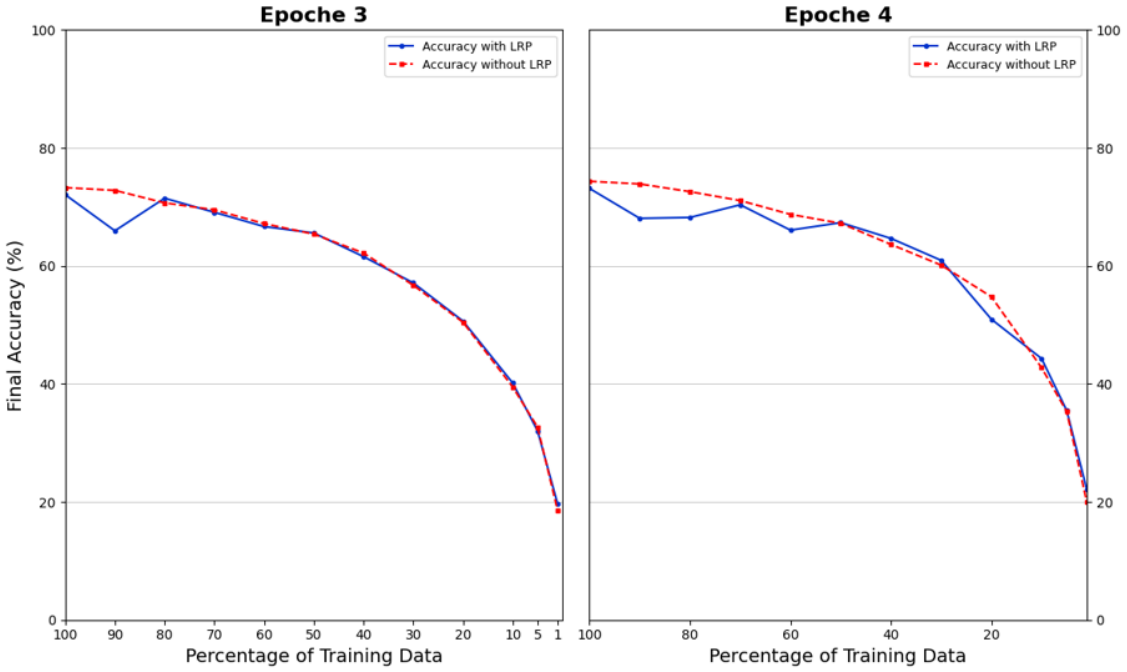

Figure 7.6: Comparison of epoch 3 and 4 with factor 0.1

When analyzing at the time needed for including LRP into training, a clear increase is caused. Table 7.2 presents the approximated extra time.

Table 7.2: Training Times with and without LRP (in seconds)

Training Data (%)	Time with LRP (s)	Time without LRP (s)
100	1654	407
90	1502	368
80	1337	332
70	1174	291
60	1008	256
50	847	215
40	680	178
30	517	140
20	359	103
10	194	66
5	111	48
1	80	33

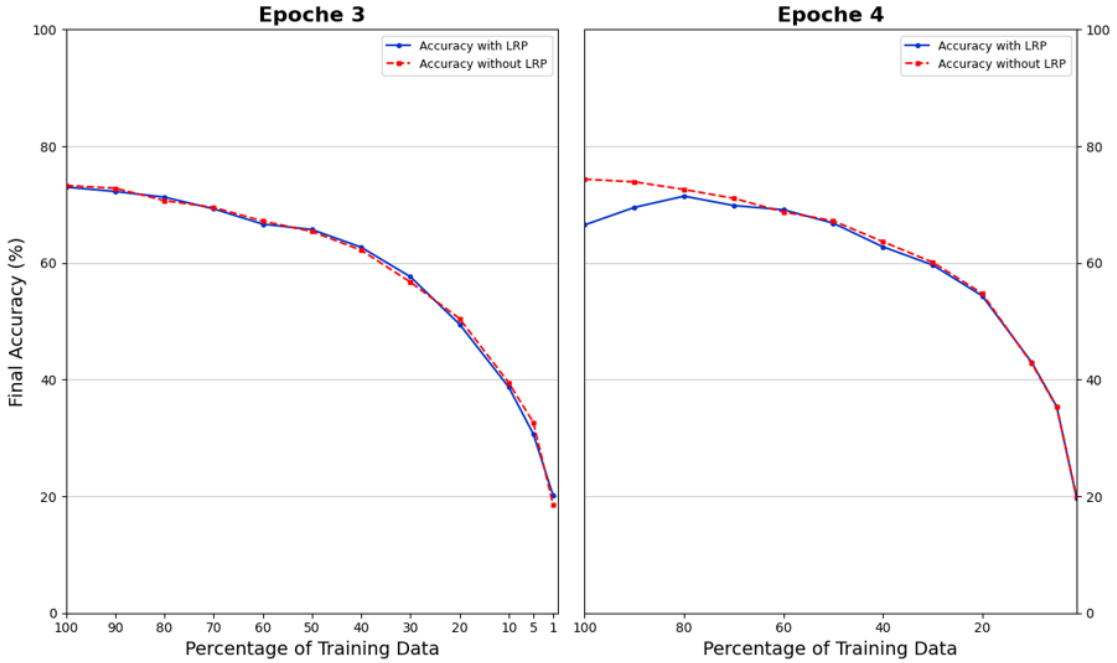

Figure 7.7: Comparison of epoch 3 and 4 with factor 0.001

7.2.2 Apply LRP only on linear part

One of the key observations from applying LRP to the model training is its effective performance when applied exclusively to the linear part of a neural network instead of the entire CNN. Figure 7.10 illustrates this approach, where only the classification part of the network is influenced by LRP.

A potential reason why LRP didn't perform well on the complete network could be due to overfitting, where the convolutional layers become overly sensitive to gradients during training. The sensitivity might result in these layers adapting excessively to the relevance adjustments, leading to instability and over-optimization. Such over-adaptation is likely to disrupt the hierarchical feature extraction process in CNNs.

The results in 7.9 visualize that the accuracy has improved after LRP was only applied to the linear part of the network. With a maximum improvement of 1.43% achieved at 50% and an average improvement of 0.93%, the improvement was consistent almost on all percentages. Also the computation time was still higher but is has decreased compared to the computation time on the whole network, as shown in 7.2.

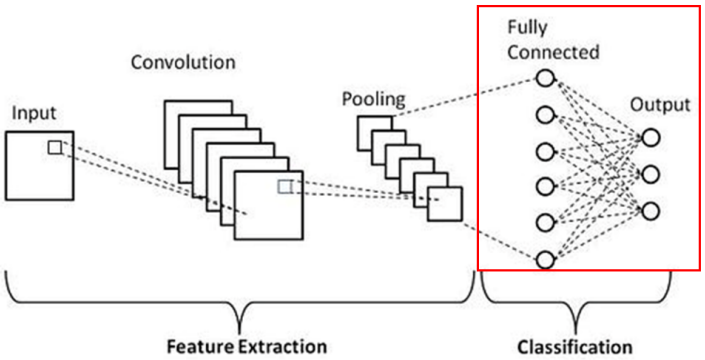

Figure 7.8: Using only the linear part [88]

Stats:
CNN Model
Number of Epochs: 3
Number of Repeats: 4
Factor: 0.1

(%)	Avg. Acc. with LRP (%)	Avg. Acc. Standard (%)	Diff
100%	74.15	73.63	✓
90%	72.84	72.24	✓
80%	71.42	70.33	✓
70%	70.25	69.83	✓
60%	68.18	67.65	✓
50%	66.18	64.75	✓
40%	61.56	61.96	✗
30%	58.46	56.88	✓
20%	49.75	50.76	✗
10%	40.64	40.21	✓
5%	34.3	33.44	✓

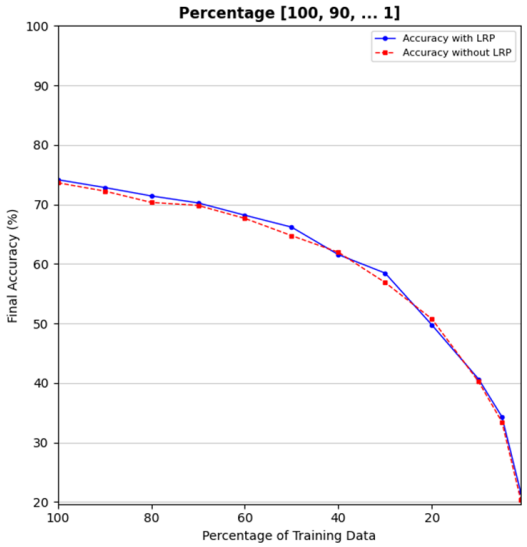

Figure 7.9: LRP only applied on linear part

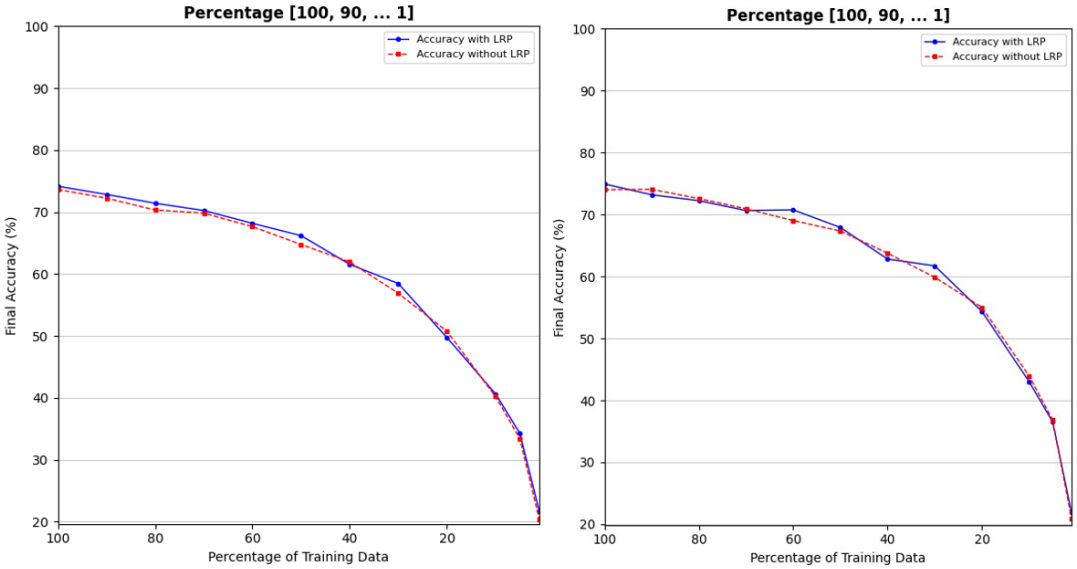

Figure 7.10: Comparison of epoch 3 and 4

Table 7.3: Training Times with and without LRP (in seconds).

Training Data (%)	Time with LRP (s)	Time without LRP (s)
100	1370	405
90	1241	370
80	1099	326
70	971	290
60	833	250
50	700	213
40	564	175
30	429	139
20	295	101
10	161	64
5	94	45

7.2.3 Integrate Early Stopping

To further improve the effect of LRP, we integrated early stopping into the training. It is a regularization technique in machine learning used to prevent overfitting during model training and works by monitoring the performance of the model on a validation dataset and stopping the training process when the validation error stops decreasing or starts increasing, which indicates that further training would lead to overfitting [89].

Traditionally, early stopping is applied epoch-wise. However, since our experiment was conducted with fewer than five epochs, a more fine-grained early stopping approach was necessary. To address this, we applied early stopping at the batch level. This means that we monitored the model's performance more frequently after specific batches rather than waiting until the end of an epoch.

Checking performance at the batch level involves evaluating the current model on the evaluation dataset, which can be computationally expensive. To mitigate this, it is impractical to test performance after every batch. Instead, we propose using a sliding window of size k, where the performance is checked after every $k - th$ batch. If the performance does not improve for l consecutive windows, the training process is stopped earlier.

Since the number of batches is determined by the dataset size divided by the batch size, the number of batches varies depending on the percentage of the dataset used. To accommodate this, both the window size k and the patience threshold l are adjusted dynamically. For higher percentages of the dataset, the window size and patience threshold are made wider, while for lower percentages, they are set to be stricter. This ensures that the early stopping mechanism remains effective and appropriately balanced across varying dataset sizes.

The results demonstrate that this batch-wise early stopping method significantly improved model performance, with an average improvement of 1.86% and a maximum improvement of 2.82%.

Stats:
CNN Model
Number of Epochs: 4
Number of Repeats: 4
Factor: 0.1

(%)	Avg. Acc. with LRP (%)	Avg. Acc. Standard (%)	Diff
100%	76.3	74.34	✓
90%	75.23	73.89	✓
80%	73.91	72.58	✓
70%	72.82	71.07	✓
60%	70.98	68.73	✓
50%	69.32	67.22	✓
40%	66.46	63.64	✓
30%	61.82	60.13	✓
20%	55.98	54.76	✓
10%	45.47	42.74	✓
5%	36.56	35.3	✓

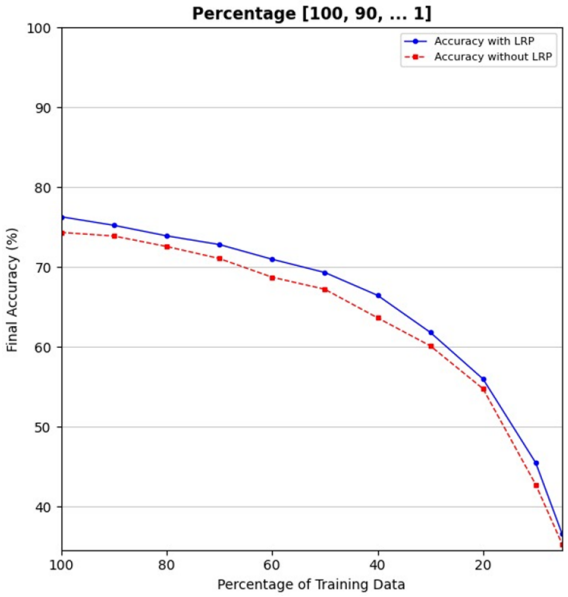

Figure 7.11: LRP only applied on linear part

Figure 7.12 from the fundamentals chapter represents how the optimal effect of enhancing LRP should ideally appear. If compared to the result with using LRP only on the linear part of the network together with early stopping, it is revealed that the approach indeed could fulfill the goal of guaranteeing the same accuracy level while using less data, proving that LRP can be indeed be used for improving model training.

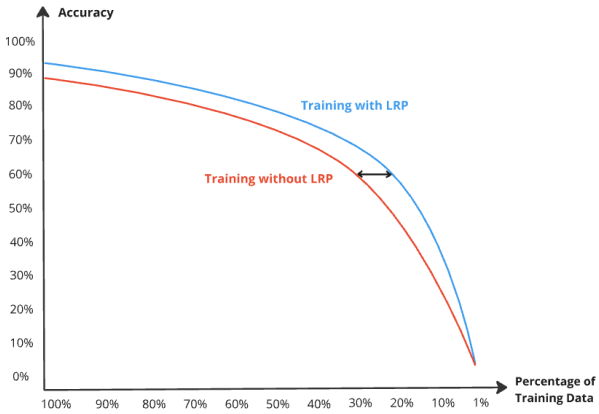

Figure 7.12: Optimal effect

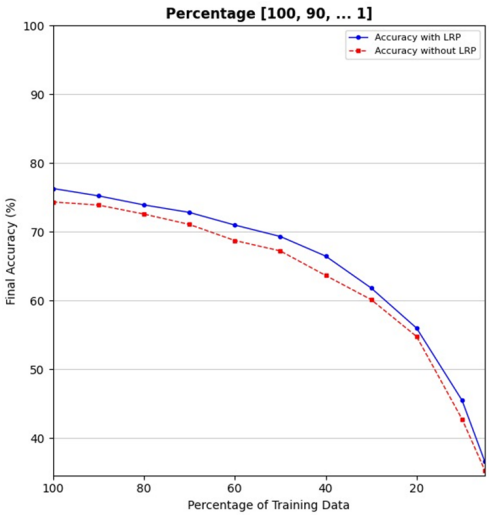

Figure 7.13: CNN only linear with early stopping

Table 7.4: Training times with and without LRP (in seconds): Early Stopping applied on Linear Layers.

Training Data (%)	Time with LRP (s)	Time without LRP (s)
100	1892.29	708.80
90	1691.20	820.93
80	1499.13	681.46
70	1293.78	626.54
60	1115.67	534.33
50	935.62	468.26
40	814.62	428.12
30	608.25	305.91
20	397.26	200.57
10	196.96	97.45
5	95.67	46.15
1	13.42	3.76

7.2.4 Apply LRP only on the whole Network

The final experiment tested the early stopping approach on the entire network, also including the convolutional layers. The results showed an improvement across almost all percentages, confirming that applying LRP in combination with early stopping has a positive impact on accuracy. Also the previously discussed overfitting issues could be prevented as the early stopping approach interrupts the training process when the accuracy suddenly drops.

Stats:
CNN Model
Number of Epochs: 3
Number of Repeats: 4
Factor: 0.1

(%)	Avg. Acc. with LRP (%)	Avg. Acc. Standard (%)	Diff
100%	76.17	75.94	✓
90%	75.46	75.03	✓
80%	74.4	73.91	✓
70%	73.18	72.63	✓
60%	71.53	71.19	✓
50%	69.82	68.98	✓
40%	66.91	65.94	✓
30%	62.49	61.28	✓
20%	55.75	55.15	✗
10%	44.28	45.57	✗

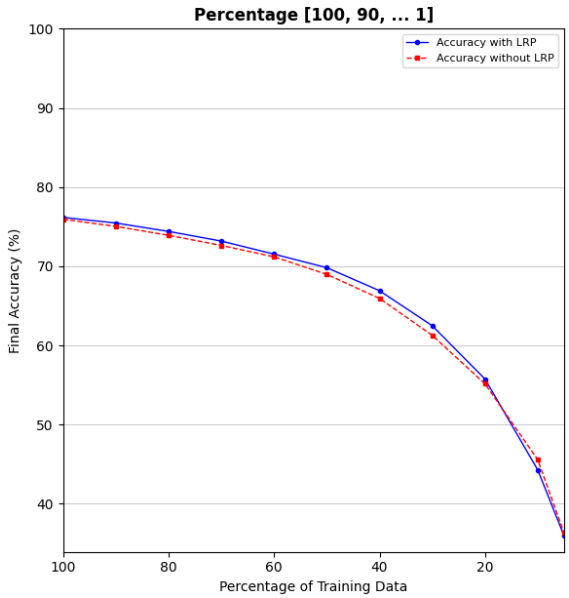

Figure 7.14: CNN early stopping on the complete network

8 Conclusion and Future Work

In this work, we explored the integration of Layer-wise Relevance Propagation into the training process of neural networks. The result of this approach showed its potential for guiding the learning process, but it also revealed several challenges and limitations that must be addressed in future research. This chapter summarizes the findings, evaluates the method's trade-offs, and discusses the opportunities for further improvement.

8.1 Linear Network and MNIST

The first experiments on the linear network together with the MNIST dataset demonstrated that integrating LRP can have a positive effect on accuracy. But only partly, as this improvement was only measurable for smaller training dataset percentages. Specifically, in the range of 1% to 30%, the results showed consistent accuracy improvement and indicated that the approach could indeed guide the model parameter updates. However, the observed improvements were too small and hardly noticeable that we cannot conclude with a substantial overall improvement. Together with that, the positive results only applied to training percentages under 30% and were not measurable when using higher percentages. Here, the opposite occurred, as issues such as gradient explosion or vanishing gradients destabilized the training process and negatively affected the training. These overfitting effects occurred more likely the bigger the LRP factor was chosen. Therefore, while the desired effect of improving accuracy was partially achieved, the overall improvement remained modest and the model was unfortunately prone to overfitting. Lastly, integrating LRP introduced additional computational time, which further reduces the practicability of the integration in this experiment if we weight it against the limited improvement it could achieve.

One reason why we could only measure limited improvements when training the linear network on MNIST could be due to the simplicity of the chosen model architecture. Although the four-layer fully connected network was suitable for the relatively simple classification of MNIST, it was likely too basic to fully harness the potential of the proposed method. Inside the model architecture, LRP was applied exclusively to the first three layers. As we explained in 6.1.5, this limitation arises because attribution values are computed between the output layer and its preceding layer but are not

applied to the output layer itself. Instead, these values are applied to the preceding layer. Starting from the output layer, relevance values are propagated backward and affect only the layers below which leaves the output layer unaffected. Therefore, the LRP enhancement was effectively restricted to just three layers and limited its scope to a small portion of the network's structure. Consequently, the network's shallow depth and restricted application of LRP made it difficult for the enhanced gradient updates to significantly improve the accuracy. A larger and more complex network addresses these limitations. The CNN from the second experiment serves as a good example. With a deeper network structure, LRP is able to influence more features and layers, which makes it easier to evaluate its potential.

Lastly, the method's effectiveness might also depend on the characteristics of the dataset. MNIST is known as a relatively simple dataset as it consists of small, low-resolution, grayscale images of handwritten digits. Because of its straightforward structure, patterns and features are relatively easy to learn using standard optimization methods. This makes it easier to achieve high accuracy even when using basic models and reduces the need for advanced techniques like LRP to guide gradient updates.

In contrast, more complex datasets, such as CIFAR-10 or ImageNet, are more difficult to classify. These datasets consist of higher-resolution images with multiple color channels and significantly more complex patterns. Due to their high dimensionality and complex decision boundaries, it is harder for models to achieve optimal performance without leveraging advanced techniques. If we apply our approach to more challenging datasets we are more likely to better demonstrate the method's potential. This could be seen in the second experiment.

8.2 Convolutional Network and CIFAR 10

Compared to the linear model and MNIST, the results of the convolutional network on the CIFAR-10 dataset provided clearer evidence of a positive effect on the model performance. When LRP was applied across the entire network first, the performance improvement was inconsistent and relatively weak. Also, accuracy drops occurred with higher percentages or increased number of epochs which indicates the same overfitting problem as with the linear network. However, a more focused application of LRP where we applied the LRP enhancement only to the fully connected layers resulted in a clearer and consistent improvement. This selective application demonstrated the potential of the method in a more complex architecture, where we achieved a consistently higher average improvement over the standard model training. The improvement was not only greater than that observed with the linear model on MNIST, but also more consistent across all the dataset percentages.

We then further improved the performance by integrating batch-wise early stopping into the training process. It monitored the training progress at the batch level and could dynamically stop training when certain performance thresholds were triggered. This helped the model to stabilize the gradients and prevent it from overfitting. With this additional modification, the method demonstrated a maximum improvement of 2.82% and an average improvement of 1.86% while applied only on the fully connected linear part. These results clearly indicate that amplifying gradient updates with attribution values effectively guide training steps and update the models into directions shaped by the computed relevance. On the contrary, the integration of early stopping increased the complexity of the training process. The new hyperparameters such as patience and window size which are used to control the early stopping process, have to be tuned and set manually. Therefore it is challenging to find a good configuration to achieve optimal results while not slowing the training process down too much.

Lastly, early stopping was also applied to the entire network, including both the convolutional and fully connected layers. The results from this experiment further validated, that guiding the gradient updates with attribution values increases the model performance. And it also approved our assumption of overfitting when LRP was applied on the whole convolutional network. By preventing the gradients to grow or to shrink to much across the whole architecture, the approach demonstrated improvements in both parts of the network. However, this setup also highlighted the inherent trade-offs. Applying early stopping together with LRP applied to the entire network improved the accuracy, but it also further increased the time to train the model.

In summary, the experiments with the convolutional network on CIFAR-10 demonstrated that LRP can positively influence training, particularly when applied selectively to fully connected layers and combined with stabilization techniques like early stopping. The additional application of early stopping to the entire network provided further insights into how relevance-based gradient guidance can enhance performance and stabilize the gradient at the same time.

8.3 Challenges and Opportunities for Improvement

8.3.1 Gradient Sensitivity

The most significant challenge of the proposed approach was that it relies highly on gradient manipulation which is inherently sensitive to changes [90]. As demonstrated in the results section, this sensitivity makes the method prone to overfitting when the LRP factor or the number of training data is high.

So far, the LRP was only a fixed single parameter that was set before training and that was applied on all neurons in the same way. And it was possible to mitigate the overfitting problem a bit by reducing the LRP factor to balance the enhancement. However, it is way too time-consuming to find the optimal integration factor and even if the optimal configuration for the factor is found, it would likely not generalize well across other models or datasets.

A solution to overcome this problem is to introduce a learnable factor that adjusts the LRP scaling factor during training. By this, the model learns the optimal scaling value to scale gradients on its own and it is not anymore necessary to manually set a fixed factor. This makes the method more flexible and stable under different training conditions. In addition, we can also use different scaling values for each of the network layers. Through this, the LRP factor is no longer a single value and is specifically trained to the unique characteristics of each layer.

For example, the learnable factor would amplify the attribution values more strongly inside deeper layers, where gradients are more prone to vanishing. On the other side, the factor would scale the attribution values less in shallower layers, where gradients are often larger, to avoid gradient explosion.

However, the implementation of learnable factors and regularization strategies introduces new challenges. For example, it increases the complexity of the model by adding additional parameters which have to be optimized. This requires careful design to ensure that the factor converges to meaningful values during training. Therefore, it is essential to conduct empirical studies to test their impact on different models and datasets in order to generalize the approach.

8.3.2 Indirect Filter Relevance

Another option to further improve the model performance could be to redefine of how LRP attribution values are computed. The current implementation of our approach computes LRP values for the convolutional layers by propagating relevance from feature map to feature map. In each of them the gradients are scaled by their corresponding attribution maps. While this approach partly works, it would be interesting to compute relevance values for the filters directly and influence their gradients directly. To clarify the difference we need to emphasize that in the current method, the gradients of the filters are only affected indirectly. They are computed through the feature maps after the gradients of the feature maps have been scaled by attribution values. A more direct approach to modifying the filter gradients might yield better results.

Directly computing relevance values for filters would align the gradient updates more closely with the actual parameters of the convolutional layers. Filters are responsible for extracting features from the input data, and their gradients determine how these features are refined during training. By influencing filter gradients directly, we can provide a more targeted adjustment based on their specific relevance and potentially improve the network's ability to learn meaningful representations.

8.3.3 Additional storage and computational costs

One of the biggest challenges of the LRP-enhanced training method, alongside the overfitting issue, is the significant increase in storage and computational requirements during training.

The storage overhead arises because the method needs to store all attribution values throughout the training process. Unlike standard training, which only temporarily stores gradients and weights during backpropagation, this approach requires saving attribution values for every neuron in fully connected layers and every feature map in the convolutional layers. This storage demand scales linearly with the number of neurons and feature maps which increases the memory demands especially in deep networks with millions of parameters. And the issue becomes even more critical when working with high-resolution datasets or architectures with many layers and filters.

In addition to the storage demands, the computational cost of this method is another critical challenge. When the approach scales the gradients with the attribution values, it requires additional calculations for each neuron or feature map during every training step. This extra operation is repeated across all layers where LRP is applied and significantly increases the overall training time.

These storage and computational costs make the practical deployment challenging. In scenarios with strict resource limits, such as embedded systems, edge devices, or cloud-based environments, the method may be infeasible. These limitations also create challenges for further research and development, where frequent experimentation and model adjustments are needed as the longer training time significantly slows down the process.

To determine whether these costs are justified, it is essential to evaluate them within the broader context and goals of the application. The primary motivation behind the LRP-enhanced method is its ability to achieve the same level of accuracy with a reduced percentage of the training dataset. If the computational cost of training with LRP is less than the time, effort or the monetary cost of obtaining and processing additional data, then the method can be considered worth it.

For example, we can consider a scenario where 20.000 data samples are available and a model has the requirement to achieve a minimum accuracy of 90%. However, the current accuracy is only 80% and it would require 10.000 additional samples more to train the model so it reaches the 90%. If using LRP enhancement allows the network to reach 90% of accuracy with only 20.000 samples instead of 30.000, the decision depends on the following trade-off. How does the cost of LRP-enhanced training compare to the cost of acquiring and processing the additional 10,000 samples? If obtaining more data is too expensive or time-consuming, then the computational demands of LRP-enhanced training could be well justified.

8.3.4 Developing a Concrete and Generalizable Strategy

The last key limitation of this study is the absence of a clear, generalizable strategy for applying LRP-enhanced training across different models and datasets. The choice of using the linear model with MNIST and the convolutional model with CIFAR-10 was intended to test the core idea of the approach. Both combinations are often used in research to test and validate new upcoming strategies and new training techniques, from which we then can generalize to larger and more complex model architectures and datasets. So although the method demonstrated partial success with MNIST and better results with CIFAR-10, its effectiveness in other contexts has still to be tried out. Especially the risk of overfitting poses a significant challenge and has to be addressed with great priority

Although we suggested a learnable factor to help the model find the optimal LRP enhancement level, developing a reliable method for determining this factor is a critical area for future research. Similarly, the early stopping strategy used in this study was only experimental and not optimized, which leaves plenty of opportunities for further hyperparameter tuning and refinement in future work.

8.4 Final Conclusion

After analyzing the results of this study, we can conclude that the LRP-enhanced training method showed potential for making training more efficient. At the same time, its practical applicability is limited by several challenges. The increased computational and storage requirements as well as the risk of drifting into overfitting during training highlight that further optimization is necessary. To unlock its full potential, it is crucial to find a more generalizable strategy that includes the previously discussed suggestions, such as dynamic LRP factors or selective application to specific layers.

By tackling these challenges, future research can improve LRP-enhanced training into a robust tool for improving neural network performance across different datasets and model architecture. Also, our approach has the potential to be especially valuable in domains where data efficiency is critical and data aggregation is expensive. Overall, this work provides a foundation for further exploration and opens the way for more effective and adaptable XAI-driven training techniques.

Bibliography

[1] Xiang Zhang, Jianxin Wu, and Hongbin Zhu. "Review of Image Classification Algorithms Based on Convolutional Neural Networks". In: *Remote Sensing* 13.22 (2021), p. 4712.

[2] Jiyoung Han et al. "Explainable Artificial Intelligence (XAI): Understanding, Visualizing and Interpreting Machine Learning Models". In: *Machine Learning and Knowledge Extraction* 3.2 (2021), pp. 320–342.

[3] Adadi. "Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)". In: *Nature Communications* (2019). DOI: 10.1038/s41467-019-08987-4. URL: https://ieeexplore.ieee.org/document/8466590/.

[4] Huawei Sun et al. "Utilizing Explainable AI for Improving the Performance of Neural Networks". In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)* (2022), pp. 2873–2879.

[5] Leander Weber et al. "Layer-wise Feedback Propagation: Enhancing Neural Network Training with Explainability". In: *arXiv preprint arXiv:2308.12053* (2023).

[6] Shiza Malik, Khalid Muhammad, and Yasir Waheed. "Artificial intelligence and industrial applications—A revolution in modern industries". In: *Ain Shams Engineering Journal* 15.5 (2024), p. 102886. DOI: 10.1016/j.asej.2024.102886. URL: https://www.researchgate.net/publication/381160287_Artificial_intelligence_and_industrial_applications-A_revolution_in_modern_industries.

[7] Vikas Hassija et al. "Interpreting Black-Box Models: A Review on Explainable Artificial Intelligence". In: *Cognitive Computation* (2023). DOI: 10.1007/s12559-023-10179-8. URL: https://link.springer.com/article/10.1007/s12559-023-10179-8.

[8] Cynthia Rudin. "Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead". In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215.

[9] Alejandro Barredo Arrieta et al. "Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI". In: *Information Fusion* 58 (2019), pp. 82–115. DOI: 10.1016/j.inffus.2019.12.012. URL: https://doi.org/10.1016/j.inffus.2019.12.012.

[10] Asharul Islam Khan and Salim Al-Habsi. "Machine Learning in Computer Vision". In: *Procedia Computer Science* 167 (2020), pp. 1444–1451. DOI: 10.1016/j.procs.2020.03.355.

[11] Metaeye. *What Are The Three Levels Of Computer Vision?* Accessed: 2024-08-01. 2024. URL: https://www.metaeye.co.uk/three-levels-of-computer-vision.

[12] ZylApp. *Review of Deep Learning Algorithms for Object Detection.* Accessed: 2024-07-13. 2023. URL: https://medium.com/zylapp/review-of-deep-learning-algorithms-for-object-detection-c1f3d437b852.

[13] Faheem Akram and Asad Abbas. "Data-driven Decisions: Exploring the Power of Machine Learning and Analytics". In: *International Journal of Data Science* 12.4 (2023), pp. 123–135. URL: https://www.researchgate.net/publication/375597596_Data-driven_Decisions_Exploring_the_Power_of_Machine_Learning_and_Analytics.

[14] I.H. Sarker. "Machine Learning: Algorithms, Real-World Applications and Research Directions". In: *SN Computer Science* 2.3 (2021), p. 160. DOI: 10.1007/s42979-021-00592-x. URL: https://doi.org/10.1007/s42979-021-00592-x.

[15] Sajid Ali et al. "Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence". In: *Information Fusion* (2023).

[16] V. V. Semerikov et al. "From Classical Machine Learning to Deep Neural Networks: A Perspective on the Development of Artificial Intelligence". In: *Applied Sciences* 11.12 (2021), p. 5541. DOI: 10.3390/app11125541. URL: https://www.mdpi.com/2076-3417/11/12/5541.

[17] Christian Janiesch, Patrick Zschech, and Kai Heinrich. "Machine learning and deep learning". In: *Electronic Markets* (2021).

[18] Frank Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.335.3398.

[19] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA: MIT Press, 1969. URL: https://mitpress.mit.edu/books/perceptrons.

[20] Aniruddha Karajgi. *How Neural Networks Solve the XOR Problem.* Accessed: 2024-07-22. 2020. URL: https://towardsdatascience.com/how-neural-networks-solve-the-xor-problem-59763136bdd7.

[21] Iqbal H. Sarker. “Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions”. In: *SN Computer Science* 2.420 (2021). DOI: 10.1007/s42979-021-00815-1. URL: https://doi.org/10.1007/s42979-021-00815-1.

[22] N. Johnson et al. *Machine Learning for Materials Developments in Metals Additive Manufacturing*. 2020. URL: https://linkinghub.elsevier.com/retrieve/pii/S2214860420310137.

[23] Ravindra Parmar. *Training Deep Neural Networks*. Accessed: 2024-08-01. 2018. URL: https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964.

[24] Leon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT* (2010), pp. 177–186.

[25] Atharva Tapkir. “A Comprehensive Overview of Gradient Descent and its Optimization Algorithms”. In: *International Advanced Research Journal in Science, Engineering and Technology* 10.11 (2023). This work is licensed under a Creative Commons Attribution 4.0 International License, pp. 37–44. ISSN: 2393-8021. DOI: 10.17148/IARJSET.2023.101106. URL: https://doi.org/10.17148/IARJSET.2023.101106.

[26] Analytics Vidhya. “Gradient Descent vs. Backpropagation: What’s the Difference?” In: (2023). URL: https://www.analyticsvidhya.com/blog/2023/01/gradient-descent-vs-backpropagation-whats-the-difference/.

[27] “Optimization Methods in Deep Learning: A Comprehensive Overview”. In: *arXiv preprint arXiv:2302.09566* (2023). URL: https://arxiv.org/pdf/2302.09566v1.

[28] Analytics Vidhya. *Stochastic Gradient Descent*. Accessed: 2024-08-06. 2024. URL: https://medium.com/analytics-vidhya/stochastic-gradient-descent-1ab661fabf89.

[29] Xiangteng He. *A Quick Guide to Gradient Descent and its Variants*. Accessed: 2024-08-06. URL: https://pub.towardsai.net/a-quick-guide-to-gradient-descent-and-its-variants-afa181d5b97b.

[30] Kumar Abhishek and Deeksha Kamath. “Attribution-based XAI Methods in Computer Vision: A Review”. In: *School of Computing Science, Simon Fraser University* (2024). † Authors contributed equally to this work. URL: mailto:kabhishe@sfu.ca,%20dkamath@sfu.ca.

[31] Mohammad Mustafa Taye. “Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions”. In: *Computation* (2023), p. 52. DOI: 10.3390/computation11030052. URL: https://www.mdpi.com/2079-3197/11/3/52.

[32] Krut Patel. *Medium*. https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9. Accessed: 2024-07-17. 2019.

[33] Fang-Xiang Wu Sakib Mostafa. *Neural Engineering Techniques for Autism Spectrum Disorder*. https://www.sciencedirect.com/book/9780128228227/neural-engineering-techniques-for-autism-spectrum-disorder. Accessed: 2024-07-17. 2021.

[34] IBM. *Convolutional Neural Networks*. Accessed: 2023-07-13. 2023. URL: https://www.ibm.com/topics/convolutional-neural-networks.

[35] Anirudha Ghosh et al. "Fundamental Concepts of Convolutional Neural Network". In: Springer, Jan. 2020, pp. 519–567. ISBN: 978-3-030-32643-2. DOI: 10.1007/978-3-030-32644-9_36.

[36] Ying Xu et al. "Convolutional Neural Networks and Feature Extraction in Image Processing for Smart Systems". In: *Procedia Computer Science* 135 (2018), pp. 335–342. DOI: 10.1016/j.procs.2018.08.187. URL: https://doi.org/10.1016/j.procs.2018.08.187.

[37] ResearchGate. *Example for the max pooling and the average pooling with a filter size of 2x2 and a stride of 2*. Accessed: 2024-07-13. 2019. URL: https://www.researchgate.net/figure/Example-for-the-max-pooling-and-the-average-pooling-with-a-filter-size-of-22-and-a_fig15_337336341.

[38] Jiuxiang Gu et al. "Recent Advances in Convolutional Neural Networks". In: *arXiv preprint arXiv:1512.07108* (2017). URL: https://arxiv.org/abs/1512.07108.

[39] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *arXiv preprint arXiv:1409.0575* (2015). URL: http://arxiv.org/abs/1409.0575.

[40] MRI Questions. *What is the Softmax Function?* Accessed: 2024-12-06. 2024. URL: https://mriquestions.com/softmax.html#/.

[41] Vaka Mohan et al. "Image Processing Representation Using Binary Image; Grayscale, Color Image, and Histogram". In: Springer India, Sept. 2016, pp. 353–361. ISBN: 978-81-322-2525-6. DOI: 10.1007/978-81-322-2526-3_37.

[42] yudam seo. *How can I plot the figure with the number of pixel as a x-axis and the grayscale color as a y-axis in python?* Accessed: 2024-08-01. 2021. URL: https://stackoverflow.com/questions/66830322/how-can-i-plot-the-figure-with-the-number-of-pixel-as-a-x-axis-and-the-grayscale.

[43] Sebastian Lapuschkin et al. "Accountability of AI Under the Law: The Role of Explanation". In: *Nature Communications* (2019).

[44] Adadi. "Concrete Problems in AI Safety". In: *Nature Communications* (2019). DOI: 10.1038/s41467-019-08987-4. URL: http://arxiv.org/abs/1606.06565.

[45] Sebastian Lapuschkin et al. “Unmasking Clever Hans Predictors and Assessing What Machines Really Learn”. In: *Nature Communications* 10.1 (2019), pp. 1–8. DOI: 10.1038/s41467-019-08987-4. URL: https://www.nature.com/articles/s41467-019-08987-4.

[46] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why should I trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (2016), pp. 1135–1144.

[47] Vaishak Belle and Ioannis Papantonis. “Principles and Practice of Explainable Machine Learning”. In: *University of Edinburgh Alan Turing Institute* (2023).

[48] Scott M Lundberg and Su-In Lee. “A Unified Approach to Interpreting Model Predictions”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, pp. 4765–4774.

[49] Madhu Ramiah. *Machine Learning Basics: Decision Trees*. Accessed: 2024-07-19. 2019. URL: https://madhuramiah.medium.com/an-introduction-to-decision-trees-ee8ac4eceab.

[50] Ian E. Nielsen et al. “Robust Explainability: A Tutorial on Gradient-Based Attribution Methods for Deep Neural Networks”. In: *ArXiv* (2021). Available at ArXiv. URL: https://arxiv.org/abs/2106.14978.

[51] Minh Dang et al. “Explainable artificial intelligence: a comprehensive review”. In: *Artificial Intelligence Review* (Nov. 2021). Accessed: 2024-07-24. DOI: 10.1007/s10462-020-09875-2. URL: https://link.springer.com/article/10.1007/s10462-020-09875-2.

[52] Carl O. Retzlaff et al. “Post-hoc vs ante-hoc explanations: xAI design guidelines for data scientists”. In: *Cognitive Systems Research* 86 (2024), p. 101243. ISSN: 1389-0417. DOI: https://doi.org/10.1016/j.cogsys.2024.101243. URL: https://www.sciencedirect.com/science/article/pii/S1389041724000378.

[53] Shahaf Bassan, Guy Amir, and Guy Katz. “Local vs. Global Interpretability: A Computational Complexity Perspective”. In: (2023). Accessed: 2024-07-23.

[54] Wenli Yang et al. “Survey on Explainable AI: From Approaches, Limitations and Applications Aspects”. In: *Human-Centric Intelligent Systems* (2023). DOI: 10.1007/s44230-023-00038-y. URL: https://link.springer.com/article/10.1007/s44230-023-00038-y.

[55] Andreas Holzinger et al. “Model-agnostic methods are universally applicable to any machine learning model”. In: *ar5iv* (2023). SpringerLink. URL: https://link.springer.com/article/10.1007/s10209-021-00795-8.

[56] Hugh Chen, Scott M. Lundberg, and Su-In Lee. “Explaining a series of models by propagating Shapley values”. In: *Nature* (2023). URL: https://www.nature.com/articles/s41467-022-31384-3.

[57] Catalina Lozano-Murcia et al. "A Comparison between Explainable Machine Learning Methods for Classification and Regression Problems in the Actuarial Context". In: *Mathematics* 11.3088 (2023). Academic Editors: Eric Ulm and Budhi Surya. DOI: 10.3390/math11143088. URL: https://www.mdpi.com/.

[58] Wojciech Samek et al. "Explaining the Decisions of Convolutional and Recurrent Neural Networks". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Springer, 2019, pp. 103–128.

[59] Jinqiang Yu, Alexey Ignatiev, and Peter J. Stuckey. "On Formal Feature Attribution and Its Approximation". In: *arXiv:2307.03380v3* (2023).

[60] Been Kim, Cynthia Rudin, and Julie A. Shah. "The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification". In: *Advances in Neural Information Processing Systems*. MIT. 2014, pp. 1952–1960.

[61] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable AI: A Review of Machine Learning Interpretability Methods". In: *Entropy* 23.1 (2021), p. 18. DOI: 10.3390/e23010018. URL: https://doi.org/10.3390/e23010018.

[62] Wojciech Samek et al. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Vol. 11700. Lecture Notes in Computer Science. Springer International Publishing, 2019. ISBN: 978-3-030-28953-9. DOI: 10.1007/978-3-030-28954-6. URL: https://link.springer.com/book/10.1007/978-3-030-28954-6.

[63] Christian Demuth. *Training Deep Neural Networks*. Accessed: 2024-08-01. 2018. URL: https://cd-anwaltskanzlei.de/templates/yootheme/cache/ef/verkehrsschild-geschwindigkeitsbegrenzung-a404139211-ef41ed71.jpeg.

[64] Emily Dineen, Rajesh Singh, and Tao Wang. "A Systematic Analysis of Perturbation-Based Explainable AI Methods". In: *arXiv preprint arXiv:2405.20200* (2024). URL: https://arxiv.org/abs/2405.20200.

[65] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning Important Features Through Propagating Activation Differences". In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR. 2017, pp. 3145–3153. URL: https://arxiv.org/abs/1704.02685.

[66] Arun Das. "Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey". In: *Nature Communications* (2019). DOI: 10.1038/s41467-019-08987-4. URL: http://arxiv.org/abs/2406.15789.

[67] Captum. *Training Deep Neural Networks*. Accessed: 2024-08-01. 2018. URL: https://captum.ai/tutorials/Image_and_Text_Classification_LIME.

[68] Lloyd S Shapley. “A Value for n-Person Games”. In: *Contributions to the Theory of Games*. Ed. by H. W. Kuhn and A. W. Tucker. Vol. 2. Princeton, NJ: Princeton University Press, 1953, pp. 307–317.

[69] Luke Merrick and Ankur Taly. “The Explanation Game: Explaining Machine Learning Models Using Shapley Values”. In: *arXiv preprint arXiv:1909.08128* (2019). URL: https://arxiv.org/abs/1909.08128.

[70] Avanti Shrikumar et al. “Learning important features through propagating activation differences”. In: *arXiv preprint arXiv:1704.02685* (2017).

[71] Yongjie Wang et al. “Gradient based Feature Attribution in Explainable AI: A Technical Review”. In: (2024).

[72] Sebastian Bach et al. “On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation”. In: *PLOS ONE* 10.7 (2015), e0130140. DOI: 10.1371/journal.pone.0130140. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140.

[73] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *preprint* (Dec. 2013).

[74] Xiangteng He. *Fine-grained Discriminative Localization via Saliency-guided Faster R-CNN - Scientific Figure on ResearchGate*. https://www.researchgate.net/figure/Some-examples-of-saliency-maps-extracted-by-SEN-in-our-Saliency-guided-Faster-R-CNN_fig4_320032994. Accessed: 2024-08-06.

[75] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[76] Jost Tobias Springenberg et al. “Striving for simplicity: The all convolutional net”. In: *International Conference on Learning Representations (ICLR) Workshop Track*. 2015. URL: https://arxiv.org/abs/1412.6806.

[77] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Not Just a Black Box: Learning Important Features Through Propagating Activation Differences”. In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. 2017. URL: https://arxiv.org/abs/1605.01713.

[78] Daniel Smilkov et al. “SmoothGrad: Removing noise by adding noise”. In: *arXiv preprint arXiv:1706.03825*. 2017. URL: https://arxiv.org/abs/1706.03825.

[79] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning (ICML)*. JMLR.org, 2017, pp. 3319–3328. URL: http://proceedings.mlr.press/v70/sundararajan17a.html.

[80] Ramprasaath R Selvaraju et al. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 618–626.

[81] Anh Pham Thi Minh. "Overview of Class Activation Maps for Visualization Explainability". In: *arXiv preprint arXiv:2309.14304* (2023). URL: https://arxiv.org/abs/2309.14304.

[82] Hugh Chen, Scott Lundberg, and Su-In Lee. "Explaining Models by Propagating Shapley Values". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. AAAI Press. 2019, pp. 12360–12367. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5050.

[83] Grégoire Montavon et al. "Layer-Wise Relevance Propagation: An Overview". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by Wojciech Samek et al. Cham: Springer, 2019, pp. 193–209. ISBN: 978-3-030-28954-6. DOI: 10.1007/978-3-030-28954-6_10.

[84] Grégoire Montavon et al. "Explaining Nonlinear Classification Decisions with Deep Taylor Decomposition". In: *Pattern Recognition* 65 (2017), pp. 211–222. DOI: 10.1016/j.patcog.2016.11.008.

[85] Li Shen et al. "On Efficient Training of Large-Scale Deep Learning Models: A Literature Review". In: *arXiv preprint arXiv:2304.03589* (2023). URL: https://arxiv.org/pdf/2304.03589.

[86] Justin Johnson and Taghi Khoshgoftaar. "The Effects of Data Sampling with Deep Learning and Highly Imbalanced Big Data". In: *Information Systems Frontiers* 22.6 (2020), pp. 1407–1423. DOI: 10.1007/s10796-020-10022-7.

[87] Sepp Hochreiter et al. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116. DOI: 10.1142/S0218488598000094. URL: https://www.researchgate.net/profile/Sepp-Hochreiter/publication/220355039_The_Vanishing_Gradient_Problem_During_Learning_Recurrent_Neural_Nets_and_Problem_Solutions/links/5700e56608ae1408e15e18e1/The-Vanishing-Gradient-Problem-During-Learning-Recurrent-Neural-Nets-and-Problem-Solutions.pdf.

[88] Edwin Dong. "Modern Convolutional Neural Network Architectures". In: *AI, But Simple* (Aug. 2024). URL: https://www.aibutsimple.com/p/modern-convolutional-neural-network-architectures.

[89] Lutz Prechelt. "Early stopping—But when?" In: *Neural Networks: Tricks of the Trade*. Springer, 1998, pp. 55–69. DOI: 10.1007/3-540-49430-8_3.

[90] Nicolas Zucchet and Antonio Orvieto. "Recurrent Neural Networks: Vanishing and Exploding Gradients Are Not the End of the Story". In: *arXiv preprint arXiv:2405.21064* (2024). URL: https://arxiv.org/abs/2405.21064.

List of Tables

List of Figures